

z/OS



MVS Programming: Authorized Assembler Services Reference, Volume 4 (SETFRR-WTOR)

z/OS



MVS Programming: Authorized Assembler Services Reference, Volume 4 (SETFRR-WTOR)

Note

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 359.

Third Edition, September 2002

This is a major revision of SA22-7612-01.

This edition applies to Version 1 Release 4 of z/OS (5694-A01), to Version 1 Release 4 of z/OS.e™ (5655-G52), and to all subsequent releases and modifications until otherwise indicated in new editions.

Order documents through your IBM® representative or the IBM branch office serving your locality. Documents are not stocked at the address below.

IBM welcomes your comments. A form for readers' comments may be provided at the back of this document, or you may address your comments to the following address:

International Business Machines Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie, NY 12601-5400
United States of America

FAX (United States & Canada): 1+845+432-9405

FAX (Other Countries):

Your International Access Code +1+845+432-9405

IBMLink™ (United States customers only): IBMUSM10(MHVRCFS)

Internet e-mail: mhvrdfs@us.ibm.com

World Wide Web: <http://www.ibm.com/servers/eserver/zseries/zos/webqs.html>

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:

- Title and order number of this document
- Page number or topic related to your comment

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© **Copyright International Business Machines Corporation 1988, 2002. All rights reserved.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures	v
Tables	vii
About this document	ix
Summary of changes	xiii
Using the Services	1
SETFRR — Set Up Functional Recovery Routines	25
SETLOCK — Control Access to Serially Reusable Resources	31
SETRP — Set Return Parameters	43
SJFREQ — Call Scheduler JCL Facility Services	53
SPIE — Specify Program Interruption Exit	87
SPOST — Synchronize POST	93
SRBSTAT — Save, Restore, or Modify SRB Status	95
SRBTIMER — Establish Time Limit for System Service	99
STATUS — Stop, Start, or Put a Subtask in Process Must-Complete Mode	103
STORAGE — Obtain and Release Storage	109
SUSPEND — Suspend Execution of an RB	127
SUSPEND — Suspend Execution of an SRB	129
SVCUPDTE — SVC Update	135
SWAREQ — Invoke SWA Manager in Locate Mode	145
SWBTUREQ — Call SJF SWBTU Processing Services	151
SYNCH and SYNCHX — Take a Synchronous Exit to a Processing Program	161
SYSEVENT — System Event	169
TCBTOKEN — Request or Translate the TTOKEN	189
TCTL — Transfer Control from an SRB Routine	197
TESTAUTH — Test Authorization of Caller	199
TIMEUSED — Obtain Accumulated CPU or Vector Time	203

T6EXIT — Type 6 Exit	207
UCBINFO — Return Information from a UCB	209
UCBLOOK — Obtain Addresses of UCB Segments	247
UCBPIN — Pinning or Unpinning a UCB	257
UCBSCAN — Scan UCBs	265
VSMLIST — List Virtual Storage Map	287
VSMLOC — Verify Virtual Storage Allocation	295
VSMREGN — Obtain Private Area Region Size	301
WAIT — Wait for One or More Events	305
WTL — Write To Log	311
WTO — Write to Operator	319
WTOR — Write to Operator with Reply.	339
Appendix. Accessibility.	357
Notices	359
Index	363

Figures

1.	Sample User Parameter List for Callers in AR Mode	5
2.	Sample Macro Syntax Diagram	13
3.	Continuation Coding	15
4.	Relationship of Data and Work Areas Referenced in IEFSJTRP	155

Tables

1. Passing User Parameters in AR Mode	4
2. Sample Callable Service Syntax Diagram	15
3. Service Summary	17
4. Return Codes for the SETLOCK Macro	35
5. Return Codes for SETLOCK RELEASE	38
6. Return Codes for SETLOCK TEST	41
7. Return and Reason Codes for the SJFREQ RETRIEVE Service	58
8. Return and Reason Codes for the SJSMBREAS Macro	62
9. Return and Reason Codes for the SJFREQ Macro SWBTU_MERGE Service	69
10. Required Fields for SJFREQ VERIFY Functions	71
11. SJFREQ VERIFY Output Fields	73
12. SJF Operand and Keyword Operand Descriptions	74
13. Return and Reason Codes for the SJFREQ Macro VERIFY Service	75
14. Return Codes from the SJFREQ TERMINATE Service	82
15. Return Codes for the SRBTIMER Macro.	101
16. Return Codes for the STATUS Macro	106
17. Return Codes for the SET/RESET Option	107
18. Return Codes for STORAGE OBTAIN.	119
19. Return Codes for STORAGE RELEASE	124
20. Return Codes for the SUSPEND Macro	131
21. Return Codes for the SVCUPDTE Macro	139
22. Return Codes for SWAREQ, UNAUTH=YES	146
23. Return Codes for the SWAREQ Macro	147
24. Parameter Combinations for SWBTUREQ RETRIEVE Functions	153
25. Return and Reason Codes for SWBTUREQ RETRIEVE	156
26. Return Codes for the SYSEVENT Macro	174
27. Example of Output of Word One from SYSEVENT STGTEST	179
28. Return Codes for REQASCL	180
29. Return Codes for REQASD and REQFASD	181
30. Return Codes for REQSRMST	182
31. Return Codes for REQLPDAT	183
32. Return Codes for ENQHOLD	184
33. Return Codes for ENQRLSE	184
34. Return Codes for QVS	187
35. Return Codes for the TCBTOKEN Macro	192
36. Return Codes for the SAMPLE Macro	202
37. Return Codes for the TIMEUSED Macro.	205
38. Return Codes for the UCBPIN Macro	261
39. Return Codes for the VSMLIST Macro	292
40. Return Codes for the VSMLOC Macro	299
41. Return and Reason Codes for the WTL Macro	313
42. MCSFLAG Flag Names	328
43. Return Codes for the WTO Macro	331
44. MCSFLAG Flag Names	346
45. Return Codes for the WTOR Macro	349

About this document

This document supports z/OS (5694–A01) and z/OS.e (5655–G52).

This document describes the authorized services that the MVS operating system provides; that is, services available only to authorized programs. An authorized program must meet one or more of the following requirements:

- Running in supervisor state
- Running under PSW key 0-7
- Residing in an APF-authorized library and link-edited with authorization code AC=1.

Some of the services included in this document are not authorized, but are included because they are of greater interest to the system programmer than to the general applications programmer. The functions of these services are of such a nature that their use should be limited to programmers who write authorized programs. Services are also included if they have one or more authorized parameters — parameters available only to authorized programs.

Programmers using assembler language can use the macros described in this document to invoke the system services that they need. This document includes the detailed information — such as the function, syntax, and parameters — needed to code the macros.

This document is divided into four volumes. Volumes 1 through 4 present the macro descriptions in alphabetical order.

Who should use this document

This document is for the programmer who is using assembler language to code a system program. A system program is usually one that runs in supervisor state or that runs with PSW key 0-7 or resides on an APF-authorized library.

The document assumes that the reader understands system concepts and writes programs in assembler language.

Assembler language programming is described in the following books:

- *HLASM Programmer's Guide*
- *HLASM Language Reference*

Using this book also requires you to be familiar with the operating system and the services that programs running under it can invoke.

How to use this document

This document is one of the set of programming documents for MVS. This set describes how to write programs in assembler language or high-level languages, such as C, FORTRAN, and COBOL. For more information about the content of this set of documents, see *z/OS Information Roadmap*.

Where to find more information

Where necessary, this document references information in other documents, using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see *z/OS Information Roadmap* (GC28-1727).

The following table lists titles and order numbers for documents related to other products.

Short Title Used in This Document	Title	Order Number
<i>Principles of Operation*</i>	<i>z/Architecture Principles of Operation</i>	SA22-7832
* Use the appropriate <i>Principles of Operation</i> document for the hardware you have installed.		
<i>PSF/MVS System Programming Guide</i>	<i>Print Services Facility/MVS System Programming Guide</i>	S544-3672

Accessing z/OS licensed documents on the Internet

z/OS licensed documentation is available on the Internet in PDF format at the IBM Resource Link™ Web site at:

<http://www.ibm.com/servers/resourceLink>

Licensed documents are available only to customers with a z/OS license. Access to these documents requires an IBM Resource Link user ID and password, and a key code. With your z/OS order you received a Memo to Licensees, (GI10-0671), that includes this key code.¹

To obtain your IBM Resource Link user ID and password, log on to:

<http://www.ibm.com/servers/resourceLink>

To register for access to the z/OS licensed documents:

1. Sign in to Resource Link using your Resource Link user ID and password.
2. Select **User Profiles** located on the left-hand navigation bar.

Note: You cannot access the z/OS licensed documents unless you have registered for access to them and received an e-mail confirmation informing you that your request has been processed.

Printed licensed documents are not available from IBM.

You can use the PDF format on either **z/OS Licensed Product Library CD-ROM** or IBM Resource Link to print licensed documents.

Using LookAt to look up message explanations

LookAt is an online facility that allows you to look up explanations for most messages you encounter, as well as for some system abends and codes. Using LookAt to find information is faster than a conventional search because in most cases LookAt goes directly to the message explanation.

You can access LookAt from the Internet at:

1. z/OS.e™ customers received a Memo to Licensees, (GI10-0684) that includes this key code.

<http://www.ibm.com/eserver/zseries/zos/bkserv/lookat/>

or from anywhere in z/OS where you can access a TSO/E command line (for example, TSO/E prompt, ISPF, z/OS UNIX System Services running OMVS). You can also download code from the *z/OS Collection* (SK3T-4269) and the LookAt Web site that will allow you to access LookAt from a handheld computer (Palm Pilot VIIx suggested).

To use LookAt as a TSO/E command, you must have LookAt installed on your host system. You can obtain the LookAt code for TSO/E from a disk on your *z/OS Collection* (SK3T-4269) or from the **News** section on the LookAt Web site.

Some messages have information in more than one document. For those messages, LookAt displays a list of documents in which the message appears.

Information updates on the web

For the latest information updates that have been provided in PTF cover letters and Documentation APARs for z/OS and z/OS.e, see the online document at:

<http://www.s390.ibm.com:80/bookmgr-cgi/bookmgr.cmd/BOOKS/ZIDOCMST/CCONTENTS>

This document is updated weekly and lists documentation changes before they are incorporated into z/OS publications.

Summary of changes

Summary of changes for SA22-7612-02 z/OS Version 1 Release 4

The document contains information previously presented in *z/OS MVS Programming: Authorized Assembler Services Reference SET-WTO*, SA22-7612-01, which supports z/OS Version 1 Release 3.

New information

- A new SYSEVENT, FREEAUX, is added.
- Information is added to indicate this books supports z/OS.e.

This document contains terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Starting with z/OS V1R2, you may notice changes in the style and structure of some content in this document—for example, headings that use uppercase for the first letter of initial words only, and procedures that have a different look and format. The changes are ongoing improvements to the consistency and retrievability of information in our documents.

Summary of changes for SA22-7612-01 z/OS Version 1 Release 3

The document contains information previously presented in *z/OS MVS Programming: Authorized Assembler Services Reference SET-WTO*, SA22-7612-00, which supports z/OS Version 1 Release 1.

New information

- A new SYSEVENT, REQLPDAT, is added.
- A new output parameter, SRMSTCAP, is added to the REQSRMST SYSEVENT.
- A new TYPE=2 keyword is added to the SYSEVENT macro, for use only by SYSEVENT ENQHOLD and ENQRLSE.
- An appendix with z/OS product accessibility information has been added.

This document contains terminology, maintenance, and editorial changes, including changes to improve consistency and retrievability.

Summary of changes for SA22-7612-00 z/OS Version 1 Release 1

The document contains information also presented in *MVS/ESA Programming: Authorized Assembler Services Reference, Volume 4 (SETFRR-WTOR)*.

New information

- A new SYSEVENT, QVS, is added.

Using the Services

Macros and callable services are programming interfaces that application programs can use to access MVS system services. This chapter provides general information and guidelines about how to use the macros and callable services accurately and efficiently. For more specific and detailed information about coding a particular macro or callable service, see the individual service description in this book.

Some of the topics covered in this chapter apply only to macros, some apply only to callable services, and some apply to both. This chapter uses the word "services" when referring to information that applies to both service types. When information applies only to one type or the other, the particular service type is specified.

The following table lists the topics covered in this chapter and whether the topic applies to macros, callable services, or both:

Topic	Service Type
"Compatibility of MVS Macros"	Macros
"Addressing Mode (AMODE)" on page 2	Both
"Address Space Control (ASC) Mode" on page 3	Both
"ALET Qualification" on page 3	Both
"User Parameters" on page 4	Macros
"Telling the System about the Execution Environment" on page 5	Macros
"Specifying a Macro Version Number" on page 6	Macros
"Register Use" on page 7	Both
"Handling Return Codes and Reason Codes" on page 8	Both
"Handling Program Errors" on page 9	Both
"Handling Environmental and System Errors" on page 9	Both
"Using X-Macros" on page 10	Macros
"Macro Forms" on page 11	Macros
"Coding the Macros" on page 12	Macros
"Coding the Callable Services" on page 15	Callable Services
"Including Equate (EQU) Statements" on page 16	Callable Services
"Link-Editing Linkage-Assist Routines" on page 16	Callable Services
"Service Summary" on page 16	Both

Compatibility of MVS Macros

When IBM introduces a new version or a new release of an existing version, the new version or release supports all MVS macros from previous versions and releases. Programs assembled on an earlier level of MVS that issue macros will run on later levels of MVS.

In most cases, the reverse is also true. When you assemble programs that issue macros on a particular version and release of MVS, those programs can run on earlier versions and releases of MVS, provided you request only those functions that are supported by the earlier version and release. This is useful for installations that write applications that might be assembled on one level of MVS, but run on a different level.

As MVS supports new architectures, addressability changes; for example, support for access registers was introduced in MVS/ESA. Support for 64-bit registers was introduced in OS/390 R10. To take best advantage of the new architectures, some macros have more than one possible expansion. You are required to have the

macro expand according to the environment in which the program runs. This topic is described in this introductory information.

The problem of compatibility is not the same as selecting a macro version via the PLISTVER parameter to ensure the correct parameter list size for a macro. For selecting a parameter list version number, see “Specifying a Macro Version Number” on page 6.

Addressing Mode (AMODE)

A program can run in 24-bit, 31-bit, or 64-bit addressing mode. A program that executes in 24-bit or 31-bit addressing mode can invoke most of the services described in this book. A program that executes in 64-bit addressing mode has a smaller group of services that it can invoke.

In general,

- A program running in 24-bit addressing mode cannot pass parameters or parameter addresses that are higher than 16 megabytes. However, there are exceptions. For example, a program running in 24-bit addressing mode can:
 - Free storage above 16 megabytes using the FREEMAIN macro
 - Allocate storage above 16 megabytes using the GETMAIN macro
 - Use cell pool services for cell pools located in storage above 16 megabytes using the CPOOL macro
 - Use page services for storage locations above 16 megabytes using the PGSER macro.
- A program running in 24-bit or 31-bit addressing mode cannot pass parameter addresses that are higher than 2 gigabytes, unless stated otherwise in the individual service description.
- If a program running in 31-bit or 64-bit addressing mode issues a service, parameters and parameter addresses can be above or below 16 megabytes, unless otherwise stated in the individual service description.

Some macros can generate code that is appropriate for programs in either 64-bit addressing mode or 24-bit or 31-bit addressing mode. These macros check a global symbol set by the SYSSTATE macro. See “Telling the System about the Execution Environment” on page 5 for more information.

When you call a callable service in 24-bit or 31-bit addressing mode, you must pass 31-bit addresses to the system service regardless of what addressing mode your program is running in. If your program is running in 24-bit mode and you use a callable service, you must set the high-order byte of parameter addresses to zeros.

You can invoke the following services in 64-bit addressing mode, subject to the “SVC or PC” restrictions mentioned below, but you may not pass parameters and parameter addresses above 2 gigabytes: ABEND, ATTACHX, CALLDISP, CHAP, CSVQUERY, DELETE, DEQ, DETACH, DOM, DSPSERV, DYNALLOC, ENQ, ESPIE, ESTAEX, EXCP, FREEMAIN, GETMAIN, IDENTIFY, GTRACE, IARVserv, LINKX, LOAD, MODESET, PHSER, POST, RESERVE, SDUMPX, SETRP, STAX, STIMER, STIMERm, STORAGE, SYNCHX, TIME, TIMEUSED, TTIMER, VRADATA, WAIT, WTO, WTOR, and XCTL.

You can invoke the following service in 64-bit addressing mode and may pass parameters and parameter addresses above 2 gigabytes: IARV64.

Before invoking a service in 64-bit addressing mode, you must inform system macros, by specifying `SYSSTATE AMODE=64`, that you are in 64-bit addressing mode. Only those options that result in calling the system by an SVC or PC may be invoked in 64-bit addressing mode. Any option that results in calling the system by a branch-entry may not be invoked in 64-bit addressing mode.

Unless explicitly stated otherwise, you should assume that a given service may not be invoked in 64-bit addressing mode and cannot accept parameters and parameter addresses above 2 gigabytes.

For information about 64-bit addressing mode and the 64-bit GPR, see *z/OS MVS Programming: Extended Addressability Guide*.

Address Space Control (ASC) Mode

A program can run in either primary ASC mode or access register (AR) ASC mode. In primary mode, the processor uses the contents of general purpose registers (GPRs) to resolve an address to a specific location. In AR mode, the processor uses the contents of ARs as well as the contents of GPRs to resolve an address to a specific location. See *z/OS MVS Programming: Assembler Services Guide* for more detailed information about AR mode.

Some macros can generate code that is appropriate for programs in either primary mode or AR mode. These macros check a global symbol set by the `SYSSTATE` macro. See “Telling the System about the Execution Environment” on page 5 for more information. Table 3 on page 17 lists the macros that check the global symbol.

Some services can generate code that is appropriate for programs in primary mode only. If you write a program in AR mode that invokes one or more services, check the description in this book for each service your program issues. Unless the description indicates that a service supports callers in AR mode, the service *does not* support callers in AR mode. In this case, use the SAC instruction to change the ASC mode of your program and issue the service in primary mode.

Whether the caller is in primary or AR ASC mode, the system uses ARs 0-1 and 14-15 as work registers across any service call.

ALET Qualification

The address space where you can place parameters varies with the individual service:

- All services allow you to place parameters in the primary address space.
- Some services *require* you to place parameters in the primary address space.
- Some services allow you to place parameters in any address space.

To identify where a service allows parameters to be located, read the individual service description.

Programs in AR mode that pass parameters must use an access register and the corresponding general purpose register together (for example, access register 1 and general purpose register 1) to identify where the parameters are located. The access register must contain an access list entry token (ALET) that identifies the address space where the parameters reside. The general purpose register must identify where, within the address space, the parameters reside.

The only ALETs that MVS services typically accept are:

- Zero (0), which specifies that the parameters reside in the caller's primary address space
- An ALET for a public entry on the caller's dispatchable unit access list (DU-AL).
- An ALET for a common area data space (CADS)

MVS services do not accept the following ALETs, and you must not attempt to pass them to a service:

- One (1), which signifies that the parameters reside in the caller's secondary address space
- An ALET that is on the caller's primary address space access list (PASN-AL) that does not represent a CADS
- An ALET for a private entry on the PASN-AL or the DU-AL.

Throughout, this book uses the term **AR/GPR *n*** to mean an access register and its corresponding general purpose register. For example, to identify access register 1 and general purpose register 1, this book uses **AR/GPR 1**.

User Parameters

Some macros that you can issue in AR mode include control parameters, user parameters, or both. Control parameters refer to the macro parameter list, and to the parameters whose addresses are in the parameter list. Control parameters control the operation of the macro itself. User parameters are parameters that the user provides to be passed through to a user routine. For example, the PARAM parameter on the ATTACHX macro defines user parameters. The ATTACHX macro passes these parameters to the routine that it attaches. All other parameters on the ATTACHX macro are control parameters that control the operation of the ATTACHX macro.

Notes:

1. User parameters are sometimes referred to as problem program parameters.
2. Control parameters are sometimes referred to as system parameters or control program parameters.

The macros shown in Table 1 allow a caller in AR mode to pass information in the form of a parameter list (or parameter lists) to another routine. This table identifies the parameter that receives the ALET-qualified address of the parameter list and tells you where the target routine finds the ALET-qualified address.

Table 1. Passing User Parameters in AR Mode

Macro	Parameter	Location of User Parameter List Address
ATTACH/ATTACHX	PARAM,VL=1	AR/GPR 1 contains the address of a list of addresses and ALETs. (See Figure 1 for the format of the list.)
ESTAEX	PARAM	SDWAPARM contains the address of an 8-byte area, which contains the address and ALET of the parameter list.

When a caller in AR mode passes ALET-qualified addresses to the called program through PARAM,VL=1 on the ATTACH/ATTACHX macro, the system builds a list formatted as shown in Figure 1 on page 5. The addresses passed to the called program are at the beginning of the list, and their associated ALETs follow the addresses. The last address in the list has the high-order bit on to indicate the size

of the list. For example, Figure 1 shows the format of a list where an AR mode issuer of ATTACHX codes the PARAM parameter as follows:

PARAM=(A,B,C),VL=1

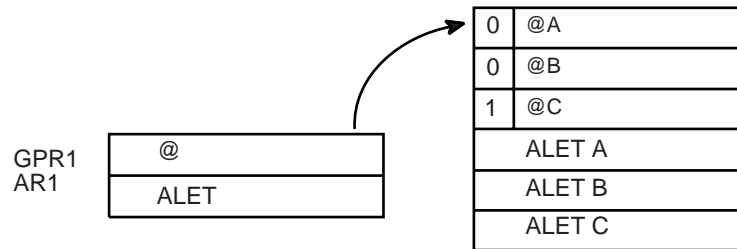


Figure 1. Sample User Parameter List for Callers in AR Mode

For information about linkage conventions, see the chapter in *z/OS MVS Programming: Assembler Services Guide*.

Telling the System about the Execution Environment

To generate code that is correct for the environment in which the program will run, some macros need to know one or more of the following characteristics about that environment:

- The addressing mode (AMODE) at the time the macro is issued
- The ASC mode of the program at the time the macro is issued
- The Architectural level in which the program will run

For macros that are sensitive to their environment, you must use the SYSSTATE macro to define the environment. During the assembly stage, SYSSTATE sets one or more global symbols. Later, when the program runs, the macro checks the global symbols and generates the correct code, which might mean avoiding use of a z/Architecture instruction or using an access register. Table 3 on page 17 lists MVS macros and identifies macros that need to know the environmental characteristics.

IBM recommends you issue the SYSSTATE macro before you issue other macros. Once a program has issued SYSSTATE, there is no need to reissue it, unless the program switches from one AMODE to another or one ASC mode to another or has code paths that are isolated according to architecture level. If you switch AMODE or ASC mode or to a different architecture code path, you should issue SYSSTATE immediately after the switch to indicate the new state. Without this information, the system assumes the macro is issued:

- In AMODE other than 64-bit
- In primary ASC mode
- In ESA/390 architectural level

The following table describes the relevant characteristics, the parameter on SYSSTATE, and the global symbol the macro checks.

Characteristic	Parameter on SYSSTATE	Global symbol
AMODE of 64-bit, or either 24-bit or 31-bit	AMODE64=YES or NO	&SYSAM64
Primary or AR ASC mode	ASCENV=P or AR	&SYSASCE

Characteristic	Parameter on SYSSTATE	Global symbol
Architectural level of: <ul style="list-style-type: none"> • ESA/390 • ESA/390 but includes the ESA/390 architecture items required by OS/390 R10 • z/Architecture 	ARCHLVL=0, 1 or 2	&SYSALVL

You can issue the SYSSTATE macro with the TEST parameter in your own user-written macro to allow your macros to generate code appropriate for their execution environment.

Callable services do not check the global symbols described in this section. To determine whether a callable service is sensitive to the AMODE, ASC mode, or the Architecture level, see the description of the individual callable service.

In early releases of MVS, the SPLEVEL macro performed a function similar to SYSSTATE. The SPLEVEL macro identifies the level of the operating system, so that a macro expansion can be tuned based on that level. This is used where macro expansions changed incompatibly. Because SPLEVEL applies to levels of the system no longer supported, it is not described in this section.

Specifying a Macro Version Number

Often there is more than one version of a macro, differentiated by additional parameters or new or expanded function. For example, version 1 of the IXGCONN macro provides connection to a log stream, while version 2 adds new parameters in support of resource manager programs. Note that this is different than using the SPLEVEL macro to select a macro version level to solve problems of downward compatibility.

You can request a specific version of a macro based on the parameters you need to use in your application, but you should also be attuned to the storage constraints of the program. The version of a macro might affect the length of the parameter list generated when the macro is assembled, because when new parameters are added to a macro, the parameter list must be large enough to fit them. The size of the parameter list might grow from release to release of OS/390 and z/OS, perhaps affecting the amount of storage your program needs.

How to Request a Macro Version Using PLISTVER

Many macros that have one or more versions supply the PLISTVER parameter. For those that do, use the PLISTVER parameter to request a version of the macro. PLISTVER is the only parameter allowed on the list form of a macro (MF), and it determines which parameter list the system generates. PLISTVER is optional. If you omit it, the system generates a parameter list for the lowest version that will accommodate the parameters specified. This is the IMPLIED_VERSION default. Note that on the list form, the default will cause the smallest parameter list to be created.

You also have the option of coding a specific version number using *plistver*, or of specifying MAX:

- *plistver* allows you to code a decimal value corresponding to the version of the macro you require. The decimal value you provide determines the amount of storage allotted for the parameter list.
- **MAX** allows you to request that the system generate a parameter list for the highest version number currently available. The amount of storage allotted for the parameter list will depend on the level of the system on which the macro is assembled.

IBM recommends, if your program can tolerate additional growth, that you always specify `PLISTVER=MAX` on the list form of the macro. `MAX` ensures that the list form parameter list is always long enough to hold whatever parameters might be specified on the execute form when both forms are assembled using the same level of the system.

Hints for Using PLISTVER

There are some general considerations that you should keep in mind when specifying the version of a macro with `PLISTVER`:

- If `PLISTVER` is omitted, the macro generates a parameter list of the lowest version that allows all the parameters specified to be processed.
- If you code `PLISTVER=n` and then specify any version '*n*+1' parameter, the macro will not assemble.
- If you code `PLISTVER=n` and do not specify any version '*n*' parameter, the macro will generate a version '*n*' parameter list.
- If you are using the standard form of the macro (`MF=S`), there is no reason you need to code the `PLISTVER` parameter.
- Not all macros in OS/390 have the same version numbers. The version numbers need not be contiguous.

The `PLISTVER` parameter appears in the syntax diagram and in the parameter descriptions. Within each macro description, the `PLISTVER` parameter description specifies the range of values and lists the parameters applicable for each version of the macro.

Register Use

Some services require that the caller place information in specific general purpose registers (GPRs) or access registers (ARs) prior to issuing the service. If a service has such a requirement, the "Input Register Information" section for the service provides that information. The section lists only those registers that have a requirement. If a register is not specified as having a requirement, then the caller does not have to place any information in that register unless using it in register notation for a particular parameter, or using it as a base register.

Once the caller issues the service, the system can change the contents of one or more registers, and leave the contents of other registers unchanged. When control returns to the caller, each register contains one of the following values or has the following status:

- The register content is preserved and is the same as it was before the service was issued.
- The register contains a value placed there by the system for the caller's use. Examples of such values are return codes and tokens.
- The system used the register as a work register. Do not assume that the register content is the same as it was before the service was issued.

Note that the system uses ARs 0, 1, 14, and 15 as work registers for every service, regardless of whether the caller is in primary or AR address space control (ASC) mode. The system does not use ARs 2 through 13 for any service.

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Many macros require that the caller have a program base register and assembler USING instruction in effect when issuing the macro; that is, the caller must have *program addressability*. AR mode programs also require that the AR associated with the caller's base GPR be set to zero. **IBM recommends** the following:

- When issuing a macro, the caller should always have program addressability in effect.
- When establishing addressability, the caller should use only registers 2 through 12.

Many macros can take advantage of relative branching when they are used with the IEABRC macro or with SYSSTATE ARCHLVL=1 or SYSSTATE ARCHLVL=2, if they are running on OS/390 version 2 release 10 or z/OS. If relative branching is used, the caller might then need addressability only to the static data portion of the program, and not to the executable code.

Handling Return Codes and Reason Codes

Most of the services described in this book provide return codes and reason codes. Return and reason codes indicate the outcome of the service in one of the following ways:

- Successful completion: you do not need to take any action.
- Successful or partially successful completion, with additional information supplied: you should evaluate the additional information in light of your particular program and determine if you need to take any action.
- Unsuccessful completion: some type of error has occurred, and you must take some action to correct the error.

The errors that cause unsuccessful completion fall into three broad categories:

Program errors	Errors that your program causes: you can correct these.
Environmental errors	Errors not caused directly by your program; rather, your program's request caused a limit to be exceeded, such as a storage limit, or the limit on the size of a particular data set. You might or might not be able to correct these.
System errors	Errors caused by the system: your program did nothing to cause the error, and you probably cannot correct these.

In some cases, a return or reason code can result from some combination of these errors.

The return and reason code descriptions for the services in this book indicate whether the error is a program error, an environmental error, a system error, or

some combination. Whenever possible, the return and reason code descriptions give you a specific action that you can take to fix the error.

IBM recommends that you read all the return and reason codes for each service that your program issues. You can then design your program to handle as many errors as possible. When designing your program, you should allow for the possibility that future releases of MVS might add new return and reason codes to a service that your program issues.

Handling Program Errors

The actions to take in the case of program errors are usually straightforward. Typical examples of program errors are:

1. Breaking one of the rules of the service. For example:
 - Passing parameters that are either in the wrong format or not valid
 - Violating one of the environment requirements (addressing mode, locking requirements, dispatchable unit mode, and so on)
 - Providing insufficient storage for information to be returned by the system.
2. Causing errors related to the parameter list. For example:
 - Coding an incorrect combination of parameters
 - Coding one or more parameters on the service incorrectly
 - Inadvertently overlaying an area of the parameter list storage
 - Inadvertently destroying the pointer to the parameter list.
3. Requesting a service or function for which the calling program is not authorized, or which is not available on the system on which the program is running.

In each of the first two cases, you can correct your program. For completeness, the return and reason code descriptions give you specific actions to perform, even when it might seem obvious what the action should be.

In the third case, you might have to contact your system administrator or system programmer to obtain the necessary authorization, or to request that the service or function be made available on your system, and the return or reason code description asks you to take that step.

Note: Generally, the system does not take dumps for errors that your program causes when issuing a system service. If you require such a dump, then it is your responsibility to request one in your recovery routine. See the section on providing recovery in *z/OS MVS Programming: Authorized Assembler Services Guide* for information about writing recovery routines.

Handling Environmental and System Errors

With environmental errors, often your first action should be to rerun your program or retry the request one or more times. The following are examples of environmental errors where rerunning your program or retrying the request is appropriate:

- The request being made through the service exceeds some internal system limit. Sometimes, rerunning your program or retrying the request results in successful completion. If the problem persists, it might be an indication of a larger problem requiring you to consult your system programmer, or possibly IBM support personnel. Your system programmer might be able to tune the system or cancel users so that the limit is no longer exceeded.
- The request exceeds an installation-defined limit. If the problem persists, the action might be to contact your system programmer and request that a specification in an installation exit or parmlib member be modified.

- The system cannot obtain storage, or some other resource, for your request. If the problem persists, the action might be to check with the operator to see if another user in the installation is causing the problem, or to see if the entire installation is experiencing storage constraint problems.

You might be able to design your program to anticipate certain environmental errors and handle them dynamically.

With system errors, as with environmental errors, often your first action should be to rerun your program or retry the request one or more times. If the problem persists, you might have to contact IBM support personnel.

Whenever possible for environmental and system errors, the return or reason code description gives you either a specific action you can take, or a list of recommended actions you can try.

For some errors, providing a specific action is not possible, because the action you should take depends on your particular application, and on what is happening in your installation. In those cases, the return or reason code description gives you one or more possible causes of the error to help you to determine what action to take.

Some system errors result in return and reason codes that are provided for IBM diagnostic purposes only. In these cases, the return or reason code description asks you to record the information and provide it to the appropriate IBM support personnel.

Using X-Macros

Some MVS services support callers in both primary and AR ASC mode. When the caller is in AR mode, macros must generate larger parameter lists; the increased size of the list reflects the addition of ALETs to qualify addresses, as described under “ALET Qualification” on page 3. For some MVS macros, two versions of a particular macro are available: one for callers in primary mode and one for callers in AR mode. The name of the macro for the AR mode caller is the same as the name of the macro for primary mode callers, except the AR mode macro name ends with an “X”. This book refers to these macros as **X-macros**.

The authorized X-macros are:

- ATTACHX
- ESTAEX
- SDUMPX
- SYNCHX

The only way these macros know that a caller is in AR mode is by checking the global symbol that the SYSSTATE macro sets. Each of these macros (and corresponding non-X-macro) checks the symbol. If SYSSTATE ASCENV=AR has been issued, the macro issues code that is valid for callers in AR mode. If it has not been issued, the macro generates code that is not valid for callers in AR mode. When your program returns to primary mode, use the SYSSTATE ASCENV=P macro to reset the global symbol.

IBM recommends that you use the X-macro regardless of whether your program is running in primary or AR mode. However, you should consider the following before deciding which macro to use:

The rules for using all X-macros, except ESTAEX, are:

- Callers in primary mode can invoke either macro.
Some parameters on the X-macros, however, are not valid for callers in primary mode. Some parameters on the non-X-macros are not valid for callers in AR mode. Check the macro descriptions for these exceptions.
- Callers in AR mode should issue the X-macros.
If a caller in AR mode issues the non-X-macro, the system substitutes the X-macro and sends a message describing the substitution.

IBM recommends you always use ESTAEX unless your program and your recovery routine are in 24-bit addressing mode, or your program requires a branch entry. In these cases, you should use ESTAE.

Macro Forms

You can code most macros in three forms: standard, list, and execute. Some macros also have a modify form. When you code a macro, you use the MF parameter to select one of the forms. The list, execute and modify forms are for reenterable programs that need to change values in the parameter list of the macro. The standard form is for programs that are not reenterable, or for programs that do not change values in the parameter list.

When a program wants to change values in the parameter list of a macro, it can make the change dynamically.

However, using the standard form and changing the parameter list dynamically might cause errors. For example, after storing a new value into the inline, standard form of the parameter list, a reenterable program operating under a given task might be interrupted by the system before the program can invoke the macro. In a multiprogramming environment, another task can use the same reenterable program, and that task might change the inline parameter list again before the first task regains control. When the first task regains control, it invokes the macro. However, the inline parameter list now has the wrong values.

Through the use of the different macro forms, a program that runs in a multiprogramming environment can avoid errors related to reenterable programs. The techniques required for using the macro forms, however, are different for some macros, called alternative list form macros, than for most other macros. For the alternative list form macros, the list form description notes that different techniques are required and refers you to the information under “Alternative List Form Macros” on page 12.

Conventional List Form Macros

With conventional list form macros, you can use the macro forms as follows:

1. Use the list form of the macro, which expands to the parameter list. Place the list form in the section of your program where you keep non-executable data, such as program constants. Do not code it in the instruction stream of your program.
2. In the instruction stream, code a GETMAIN or a STORAGE macro to obtain some virtual storage.
3. Code a move character instruction that moves the parameter list from its non-executable position in your program into the virtual storage area that you obtained.

4. For macros that have a modify form, you can code the modify form of the macro to change the parameter list. Use the address parameter of the modify form to reference the parameter list in the virtual storage area that you obtained. Thus, the parameter list that you change is the one in the virtual storage area obtained by the GETMAIN or STORAGE macro.
5. Invoke the macro by issuing the execute form of the macro. Use the address parameter of the execute form to reference the parameter list in the virtual storage area that you obtained.

With this technique, the parameter list is safe even if the first task is interrupted and a second task intervenes. When the program runs under the second task, it cannot access the parameter list in the virtual storage of the first task.

Alternative List Form Macros

Certain macros, called alternative list form macros, require a somewhat different technique for using the list form. With these macros, you do not move the area defined by the list form into virtual storage that you have obtained; instead, you place the area defined by the list form into a DSECT. Also, it is the list form, not the execute form, that you use to specify the address parameter that identifies the address of the storage for the parameter list. Note that no modify form is available for these macros.

You can use the macro forms for the alternative list form macros as follows:

1. Use the list form of the macro to define an area of storage that the execute form can use to store the parameters. As with other macros, do not code the list form in the instruction stream of your program.
2. In the instruction stream, code a GETMAIN or a STORAGE macro to obtain virtual storage for the list form expansion.
3. Place the area defined by the list form into a DSECT that maps a portion of the virtual storage you obtained.
4. Invoke the macro by issuing the execute form of the macro. The address parameter specified on the list form references the parameter list in the virtual storage area that you obtained.

Coding the Macros

In this book, each macro description includes a syntax table near the beginning of the macro description. The table shows how to code the macro. The syntax table does not explain the meanings of the parameters; the meanings are explained in the parameter descriptions that follow the syntax table.

The syntax tables assume that the standard begin, end, and continue columns are used. Thus, column 1 is assumed as the begin column. To change the begin, end, and continue columns, use the ICTL instruction to establish the coding format you want to use. If you do not use ICTL, the assembler recognizes the standard columns. To code the ICTL instruction, see *HLASM Language Reference*.

Figure 2 shows a sample macro, TEST, and summarizes all the coding information that is available for it. The table is divided into three columns, A, B, and C.

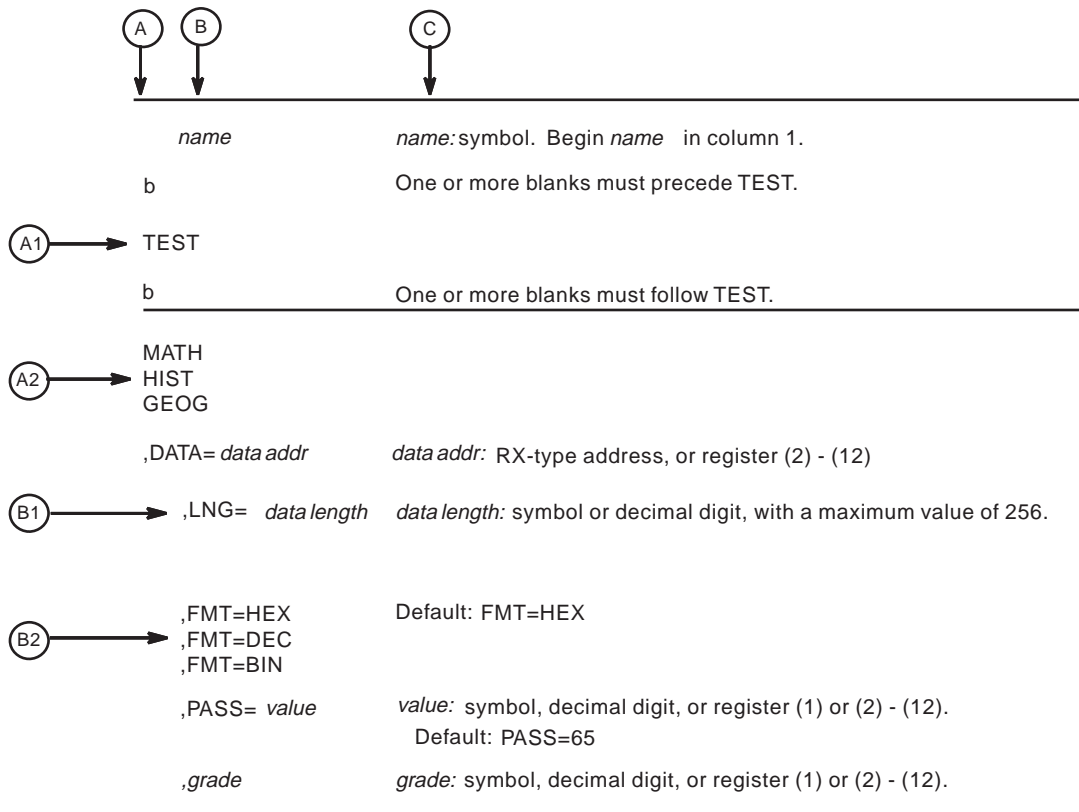


Figure 2. Sample Macro Syntax Diagram

- Column A and Column B contain those parameters that are allowed for the macro. Column A contains those parameters that are required; column B contains those parameters which are optional.
- If a single line appears, as shown in A1 and B1, then that is the only available choice for the particular parameter.
- If two or more lines appear together, as shown in A2 and B2, the parameters on those lines are mutually exclusive, that is, you can code any one of those parameters.
- A further distinction is made between mandatory and optional parameters. The parameter descriptions that follow the syntax table clearly identify those parameters which are optional.
- The third column, C, provides additional information about coding the macro.

When substitution of a variable is required in column C, the following classifications are used:

Variable	Classification
<i>Symbol</i>	Any symbol valid in the assembler language. The symbol can be as long as the supported maximum length of a name entry in the assembler you are using.
<i>Decimal digit</i>	Any decimal digit up to and including the value indicated in the parameter description. If both symbol and decimal digit are indicated, an absolute expression is also allowed.

Register (2)-(12)

One of general purpose registers 2 through 12, specified within parentheses, previously loaded with the right-adjusted value or address indicated in the parameter description. You must set the unused high-order bits to zero. You can designate the register symbolically or with an absolute expression.

Register (0)

General purpose register 0, previously loaded with the right-adjusted value or address indicated in the parameter description. You must set the unused high-order bits to zero. Designate the register as (0) only.

Register (1)

General purpose register 1, previously loaded with the right-adjusted value or address indicated in the parameter description. You must set the unused high-order bits to zero. Designate the register as (1) only.

Register (15)

General purpose register 15, previously loaded with the right-adjusted value or address indicated in the parameter description. You must set the unused high-order bits to zero. Designate the register as (15) only.

RX-type address

Any address that is valid in an RX-type instruction (for example, LA).

RS-type address

Any address that is valid in an RS-type instruction (for example, STM).

RS-type name

Any name that is valid in an RS-type instruction (for example, STM).

A-type address

Any address that can be written in an A-type address constant.

Default

A value that is used in default of a specified value; that is, the value the system assumes if the parameter is not coded.

Use the parameters to specify the services and options to be performed, and write them according to the following rules:

- If the selected parameter is written in all capital letters (for example, MATH, HIST, or FMT=HEX), code the parameter exactly as shown.
- If the selected parameter is written in italics (for example, *grade*), substitute the indicated value, address, or name.
- If the selected parameter is a combination of capital letters and italics separated by an equal sign (for example, DATA=*data addr*), code the capital letters and equal sign as shown, and then make the indicated substitution for the italics.
- Read the table from top to bottom.
- Code commas and parentheses exactly as shown.
- Positional parameters (parameters without equal signs) appear first; you must code them in the order shown. You may code keyword parameters (parameters with equal signs) in any order.
- If you select a parameter, read the third column before proceeding to the next parameter. The third column often contains coding restrictions for the parameter.

Continuation Lines

You can continue the parameter field of a macro on one or more additional lines according to the following rules:

- Enter a continuation character (not blank, and not part of the parameter coding) in column 72 of the line.
- Continue the parameter field on the next line, starting in column 16. All columns to the left of column 16 must be blank.

You can code the parameter field being continued in one of two ways. Code the parameter field through column 71, with no blanks, and continue in column 16 of the next line; or truncate the parameter field by a comma, where a comma normally falls, with at least one blank before column 71, and then continue in column 16 of the next line. Figure 3 shows an example of each method.

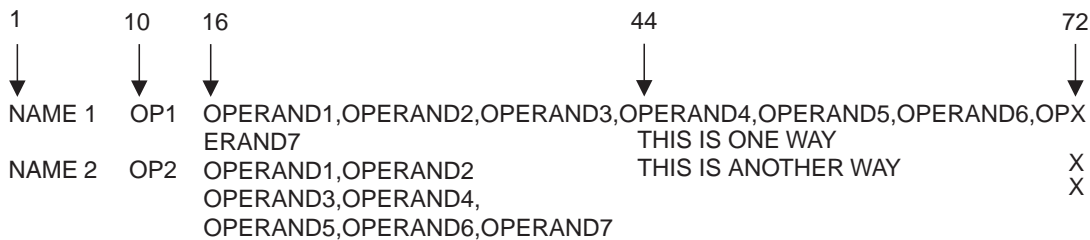


Figure 3. Continuation Coding

Coding the Callable Services

A callable service is a programming interface that uses the CALL macro to access system services. To code a callable service, code the CALL macro followed by the name of the callable service, and a parameter list; for example:

```
CALL service,(parameter list)
```

Table 2 shows the syntax diagram for the sample callable service SCORE.

Table 2. Sample Callable Service Syntax Diagram

CALL SCORE	,(test_type ,level ,data ,format_option ,return_code)
------------	---

Considerations for coding callable services are:

- You must code all the parameters in the parameter list because parameters are positional in a callable service interface. That is, the function of each parameter is determined by its position with respect to the other parameters in the list. Omitting a parameter, therefore, assigns the omitted parameter's function to the next parameter in the list.
- You must place values explicitly into all input parameters, because callable services do not set default values.

- You can use the list and execute forms of the CALL macro to preserve your program's reentrancy.

Including Equate (EQU) Statements

IBM supplies sets of equate (EQU) statements for use with some callable services. These statements, which you may optionally include in your source code, provide constants for use in your program. IBM provides the statements as a programming convenience to save you the trouble of coding the definitions yourself.

Note: Check the "Programming Requirements" section of the individual service description to determine if the equate statements are available for the callable service you are using. If the equate statements are available, that section will also provide a list of the statements that are provided, along with a description of how to include them in your program.

Link-Editing Linkage-Assist Routines

Linkage-assist routines provide the connection between your program and the system services that your program requests. When using callable services, link-edit the appropriate linkage-assist routines into your program module so that, during execution, the linkage-assist routines can resolve the address of, and pass control to, the requested system services. You can also dynamically link to linkage-assist routines as an alternative to link-editing. For example, issue the LOAD macro for the linkage-assist routine, then issue a CALL to the loaded addresses.

To invoke the linkage-editor or binder, code JCL as in the following example:

```
//userid JOB 'accounting-info','name',CLASS=x,
// MSGCLASS=x,NOTIFY=userid,MSGLEVEL=(1,1),REGION=4096K
//LINKSTEP EXEC PGM=HEWL,
// PARM='LIST,LET,XREF,REFR,RENT'
//SYSPRINT DD SYSOUT=x
//SYSLMOD DD DSN=userid.LOADLIB,DISP=OLD
//SYSLIB DD DSN=SYS1.CSSLIB,DISP=SHR
//OBJLIB DD DSN=userid.OBJLIB,DISP=SHR
//SYSUT1 DD UNIT=SYSDA,SPACE=(TRK,(5,2))
//SYSLIN DD *
INCLUDE OBJLIB(userpgm)
ENTRY userpgm
NAME userpgm(R)
/*
```

Note: Omitting NCAL from the linkedit parameters (as the example shows) and specifying SYS1.CSSLIB in the //SYSLIB statement, as shown, causes the addresses of all required linkage-assist routines to be automatically resolved. This statement saves you the trouble of having to specify individual linkage-assist routines in INCLUDE statements.

Service Summary

Table 3 on page 17 lists services described in the following:

- *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN*
- *z/OS MVS Programming: Authorized Assembler Services Reference ENF-IXG*
- *z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDU*
- *z/OS MVS Programming: Authorized Assembler Services Reference SET-WTO.*

For each service, the table indicates:

- Whether a program in AR ASC mode can issue the service

- Whether a program in cross memory mode can issue the service
- Whether the macro checks the SYSSTATE global variable
- Whether the macro can be issued in 64-bit addressing mode.

Notes:

1. A program running in primary ASC mode when PASN=SASN=HASN can issue any of the services listed in the table.

2. Cross memory mode means that at least one of the following conditions is true:

PASN \neq SASN	The primary address space (PASN) and the secondary address space (SASN) are different.
PASN \neq HASN	The primary address space (PASN) and the home address space (HASN) are different.
SASN \neq HASN	The secondary address space (SASN) and the home address space (HASN) are different.

For more information about functions that are available to programs in cross memory mode, see *z/OS MVS Programming: Extended Addressability Guide*.

3. Callable services do not check the SYSSTATE or SPLEVEL global variables.

Table 3. Service Summary

Service	Can be issued in AR ASC mode	Can be issued in cross memory mode	Checks SYSSTATE	Can be issued in 64-bit AMODE
ALESERV	Yes	Yes	No	No
ASCRE	Yes	Yes	Yes	No
ASDES	Yes	Yes	Yes	No
ASEXT	Yes	Yes	No	No
ATSET	No	Yes	No	No
ATTACH	Yes (See note 1 on page 24)	No	Yes	No
ATTACHX	Yes	No	Yes	Yes
AXEXT	No	Yes	No	No
AXFRE	No	Yes	No	No
AXRES	No	Yes	No	No
AXSET	No	Yes	No	No
BPXEKDA	Yes	No	Yes	No
BPXESMF	Yes	No	Yes	No
CALLDISP	No	Yes	No	Yes
CALLRTM	No	Yes (See note 2 on page 24)	No	No
CHANGKEY	No	Yes	No	No
CIRB	No	No	No	No
CMDAUTH	No	No	No	No
COFCREAT	Yes	Yes	Yes	No
COFDEFIN	Yes	Yes	Yes	No
COFIDENT	Yes	Yes	Yes	No
COFNOTIF	Yes	Yes	Yes	No

Table 3. Service Summary (continued)

Service	Can be issued in AR ASC mode	Can be issued in cross memory mode	Checks SYSSTATE	Can be issued in 64-bit AMODE
COFPURGE	Yes	Yes	Yes	No
COFREMOV	Yes	Yes	Yes	No
COFRETRI	Yes	Yes	Yes	No
COFSDONO	No	No	Yes	No
CONFCHG	No	No	Yes	No
CPF	No	No	No	No
CPOOL	No	Yes	No	No
CSRSI	No	Yes	No	No
CSRUNIC	Yes	Yes	No	No
CSVAPF	Yes (See note 11 on page 24)	Yes (See note 12 on page 24)	Yes	No
CSVODYNEX	Yes (See note 13 on page 24)	Yes (See note 14 on page 24)	Yes	No
CTRACE	No	No	Yes	No
CTRACECS	Yes	No	Yes	No
CTRACEWR	Yes	Yes	Yes	No
DATOFF	Yes	No	No	No
DEQ	No	No	No	Yes
DIV	Yes	No	Yes	No
DOM	No	No	No	Yes
DSPSERV	Yes	Yes	Yes	Yes
DYNALLOC	No	No	No	Yes
ENFREQ	No	No	No	No
ENQ	No	No	No	Yes
ESPIE	No	No	No	Yes
ESTAE (See note 3 on page 24)	No	No	Yes	No
ESTAEX	Yes	Yes	Yes	Yes
ETCON	No	Yes	No	No
ETCRE	No	Yes	No	No
ETDEF	Yes	Yes	No	No
ETDES	No	Yes	No	No
ETDIS	No	Yes	No	No
EVENTS	No	No	No	No
EXTRACT	No	No	No	No
FESTAE	No	No	No	No
FREEMAIN	Yes (See note 4 on page 24)	Yes	Yes	Yes
GETDSAB	No	No	Yes	No

Table 3. Service Summary (continued)

Service	Can be issued in AR ASC mode	Can be issued in cross memory mode	Checks SYSSTATE	Can be issued in 64-bit AMODE
GETMAIN	Yes (See note 4 on page 24)	Yes	Yes	Yes
GQSCAN	No	Yes	No	No
GTRACE	No	Yes	No	Yes
HSPSERV	Yes	Yes (See note 5 on page 24)	(See note 6 on page 24)	No
IARR2V	Yes	Yes	No	No
IARSUBSP	Yes	Yes	Yes	No
IARVSERV	Yes	Yes	Yes	No
IARV64	Yes	Yes	Yes	Yes
IAZXJSAB	Yes	Yes (See note 15 on page 24)	Yes	No
IEAARR	Yes	Yes	Yes	No
IEALSQRY	Yes	Yes	Yes	No
IEAMRMF3	No	Yes	No	No
IEAMSCHD	Yes	Yes	Yes	No
IEANTCR	Yes	Yes	N/A	No
IEANTDL	Yes	Yes	N/A	No
IEANTRT	Yes	Yes	N/A	No
IEARBUP	Yes	Yes	Yes	No
IEATDUMP	Yes	No	Yes	No
IEAVAPE	No	Yes	No	No
IEAVDPE	No	Yes	No	No
IEAVPSE	No	Yes	No	No
IEAVRLS	No	Yes	No	No
IEAVRPI	No	Yes	No	No
IEAVTPE	No	Yes	No	No
IEAVXFR	No	Yes	No	No
IEEQEMCS	Yes	Yes	Yes	No
IEEVARYD	No	No	Yes	No
IEFPPSCN	No	No	Yes	No
IEFQMREQ	No	No	No	No
IEFSSI	Yes	No	No	No
IEFSSVT	Yes	No	No	No
IEFSSVTI	Yes	Yes	No	No
IOSADMF	No	No	Yes	No
IOSCAPF	No	Yes (See note 7 on page 24)	Yes	No
IOSCAPU	Yes	Yes (See note 7 on page 24)	Yes	No

Table 3. Service Summary (continued)

Service	Can be issued in AR ASC mode	Can be issued in cross memory mode	Checks SYSSTATE	Can be issued in 64-bit AMODE
IOSCDR	No	No	Yes	No
IOSCHPD	Yes	Yes	Yes	No
IOSCMXA	No	Yes (See note 7 on page 24)	Yes	No
IOSCMXR	No	Yes (See note 7 on page 24)	Yes	No
IOSDCXR	No	Yes (See note 7 on page 24)	Yes	No
IOSENQ	Yes	Yes	Yes	No
IOSINFO	No	No	No	No
IOSLOOK	No	No	No	No
IOSPTHV	No	No	Yes	No
IOSUPFA	No	Yes	Yes	No
IOSUPFR	No	Yes	Yes	No
IOSWITCH	Yes	Yes	Yes	No
ISGLCRT	No	Yes	N/A	No
ISGLOBT	No	Yes	N/A	No
ISGLREL	No	Yes	N/A	No
ISGLPRG	No	Yes	N/A	No
ITTFMTB	No	No	No	No
ITZXFLT	No	Yes	Yes	No
IWMCLSFY	No	Yes	Yes	No
IWMCONN	No	Yes	Yes	No
IWMDISC	No	Yes	Yes	No
IWMECQRY	No	Yes	Yes	No
IWMECREA	No	Yes	Yes	No
IWMEDELE	No	Yes	Yes	No
IWMMABNL	No	Yes	No	No
IWMMCHST	No	Yes	No	No
IWMMCREA	No	Yes	Yes	No
IWMMDELE	No	Yes	Yes	No
IWMMEXTR	No	Yes	Yes	No
IWMMINIT	No	Yes	No	No
IWMMNTFY	No	Yes	Yes	No
IWMMRELA	No	Yes	Yes	No
IWMMSWCH	No	Yes	Yes	No
IWMMXFER	No	Yes	No	No
IWMPQRY	Yes	Yes	Yes	No
IWMRCOLL	Yes	Yes	Yes	No

Table 3. Service Summary (continued)

Service	Can be issued in AR ASC mode	Can be issued in cross memory mode	Checks SYSSTATE	Can be issued in 64-bit AMODE
IWMRPT	No	Yes	Yes	No
IWMRQRY	Yes	Yes	Yes	No
IWMSRDRS	No	Yes	Yes	No
IWMSRSRG	No	Yes	Yes	No
IWMSRSRS	No	Yes	Yes	No
IWMWMCON	No	Yes	Yes	No
IWMWQRY	Yes	Yes	Yes	No
IWMWQWRK	No	Yes	Yes	No
IXCCREAT	Yes	Yes	Yes	No
IXCDELET	Yes	Yes	Yes	No
IXCJOIN	Yes	No	Yes	No
IXCLEAVE	Yes	No	Yes	No
IXCMG	Yes	Yes	Yes	No
IXCMOD	Yes	Yes	Yes	No
IXCMSGI	Yes	No	Yes	No
IXCMSGO	Yes	Yes	Yes	No
IXCQUERY	Yes	Yes	Yes	No
IXCQUIES	Yes	No	Yes	No
IXCSETUS	Yes	Yes	Yes	No
IXCTERM	Yes	Yes	Yes	No
LLACOPY	No	No	Yes	No
LOAD	Yes	No	No	Yes
LOADWAIT	No	Yes	Yes	No
LOCASCB	Yes	Yes	Yes	No
LXFRE	No	Yes	No	No
LXRES	No	Yes	No	No
MCSOPER	Yes	No	Yes	No
MCSOPMSG	Yes	No	Yes	No
MGCR	No	No	No	No
MGCRE	No	No	No	No
MIHQUERY	Yes	No	Yes	No
MODESET	No	Yes	No	Yes
NIL	No	No	No	No
NMLDEF	No	No	No	No
NUCLKUP	No	No	No	No
OIL	No	No	No	No
OUTADD	No	No	No	No
OUTDEL	No	No	No	No

Table 3. Service Summary (continued)

Service	Can be issued in AR ASC mode	Can be issued in cross memory mode	Checks SYSSTATE	Can be issued in 64-bit AMODE
PCLINK	No	Yes	No	No
PGANY	No	No	No	No
PGFIX	No	Yes	No	No
PGFIXA	No	No	No	No
PGFREE	No	Yes	No	No
PGFREEA	No	No	No	No
PGSER	Yes (See note 8 on page 24)	Yes (See note 8 on page 24)	No	Yes
POST	No	Yes	No	Yes
PTRACE	No	Yes	No	No
PURGEDQ	No	No	No	No
QEDIT	No	No	No	No
RESERVE	No	No	No	Yes
RESMGR	Yes	Yes	No	No
RESUME	No	Yes	No	No
RISGNL	No	Yes	No	No
SCHEDIRB	Yes	No	Yes	No
SCHEDULE	Yes	Yes	Yes	No
SCHEDXIT	No	Yes	No	No
SDUMP	Yes (See note 1 on page 24)	Yes (See note 9 on page 24)	Yes	No
SDUMPX	Yes	Yes (See note 9 on page 24)	Yes	Yes
SETFRR	Yes	Yes	Yes	No
SETLOCK	Yes	Yes	Yes	No
SETRP	Yes	Yes	Yes	Yes
SJFREQ	No	Yes	No	No
SPIE	No	No	No	No
SPOST	No	No	No	No
SRBSTAT	No	Yes	No	No
SRBTIMER	No	No	No	No
STATUS	Yes	Yes	No	No
STORAGE	Yes	Yes	No	Yes
SUSPEND	No	Yes	No	No
SVCUPDTE	No	No	No	No
SWAREQ	No	No	No	No
SWBTUREQ	No	No	No	No
SYNCH	Yes (See note 1 on page 24)	No	Yes	No
SYNCHX	Yes	No	Yes	Yes

Table 3. Service Summary (continued)

Service	Can be issued in AR ASC mode	Can be issued in cross memory mode	Checks SYSSTATE	Can be issued in 64-bit AMODE
SYSEVENT	No	No	No	No
TCBTOKEN	Yes	Yes	No	No
TCTL	No	No	No	No
TESTAUTH	No	No	No	No
TIMEUSED	Yes (See note 10 on page 24)	Yes	No	Yes
T6EXIT	No	No	No	No
UCBINFO	Yes	Yes	Yes	No
UCBLOOK	Yes	Yes	Yes	No
UCBPIN	Yes	Yes	Yes	No
UCBSCAN	Yes	Yes	Yes	No
VSMLIST	No	Yes	No	No
VSMLOC	No	Yes	No	No
VSMREGN	No	Yes	No	No
WAIT	No	Yes	No	Yes
WTL	No	No	No	No
WTO	No	No	No	Yes
WTOR	No	No	No	Yes

Table 3. Service Summary (continued)

Service	Can be issued in AR ASC mode	Can be issued in cross memory mode	Checks SYSSTATE	Can be issued in 64-bit AMODE
<p>Notes:</p> <ol style="list-style-type: none"> Primary mode callers can use either macro in the following macro pairs: ATTACH or ATTACHX SDUMP or SDUMPX SYNCH or SYNCHX IBM recommends that programs in AR ASC mode use the X-macros (ATTACHX, SDUMPX, and SYNCHX). If, however, a program in AR mode issues ATTACH, SDUMP, or SYNCH after issuing SYSSTATE ASCENV=AR, the system substitutes the corresponding X-macro and issues a message telling you that it made the substitution. CALLRTM TYPE=MEMTERM can be issued in cross memory mode. For CALLRTM TYPE=ABTERM, see the CALLRTM macro description. The only programs that can use ESTAE are programs that are in primary mode with (PASN=SASN=HASN). IBM recommends you always use ESTAEX unless your program and your recovery routine are in 24-bit addressing mode, or your program requires a branch entry. In these cases, you should use ESTAE. IBM recommends that AR mode callers use the STORAGE macro instead of using GETMAIN or FREEMAIN. For HSPSERV SREAD and HSPSERV SWRITE, PASN=HASN=SASN for a non-shared standard hiperspace for which an ALET is not used (that is, the HSPALET parameter is omitted). If you use the HSPALET parameter, the HSPSERV macro checks SYSSTATE. If the input UCB is captured, the IOSCAPF, IOSCMXA, IOSCMXR, and IOSDCXR macros can be issued in cross memory mode only if the UCB is captured in the primary address space. IOSCAPU CAPTOACT without the ASID parameter also can be issued in cross memory mode if the UCB was captured in the primary address space. IOSCAPU CAPTUCB and IOSCAPU UCAPTUCB cannot be issued in cross memory mode. PGSER can be issued in AR ASC mode only if you specify BRANCH=Y. PGSER can be issued in cross memory mode only if you specify BRANCH=Y or BRANCH=SPECIAL. Both SDUMP and SDUMPX can be issued in cross memory mode only if you specify BRANCH=YES. Only TIMEUSED LINKAGE=SYSTEM can be issued in AR ASC mode. TIMEUSED LINKAGE=BRANCH cannot be issued in AR ASC mode. For a QUERY request, CSVAPF can be issued only in primary mode. For all other requests, CSVAPF can be issued in primary or AR mode. For CSVAPF with the ADD, DELETE, and DYNFORMAT requests, PASN = HASN = SASN. For CSVAPF with the QUERY, QUERYFORMAT, and LIST requests, any PASN, any HASN, any SASN. For a QUERY or a CALL request with FASTPATH=YES, CSVDYNEX can be issued only in primary mode. For all other requests, CSVDYNEX can be issued in primary or AR mode. For CSVDYNEX CALL, RECOVER, and QUERY requests, any PASN, any HASN, any SASN. For all other requests, PASN=HASN=SASN. When the caller of the IAZXJSAB macro specifies the ASCB parameter, any PASN, any HASN, any SASN; otherwise, PASN=HASN is required. 				

SETFRR — Set Up Functional Recovery Routines

Description

The SETFRR macro gives authorized programs the ability to define their recovery in the FRR (functional recovery routine) LIFO stack, which is used during processing of the system recovery manager. Any program function can use SETFRR to define its own unique recovery environment.

The SETFRR macro can be used to add, delete, or replace FRRs in the LIFO stack, or to purge all FRRs in the stack. The macro also optionally returns to the user the address of a parameter area that is eventually passed to the FRR when an error occurs. The parameter area can be used to keep information that might be useful to the FRR. The recovery and retry routines execute in the same addressing mode as the issuer of the macro.

z/OS MVS Programming: Authorized Assembler Services Guide describes the interface to an FRR and contains guidelines for writing an FRR.

Environment

The requirements for the caller are:

Minimum authorization:	Supervisor state and PSW key 0
Dispatchable unit mode:	Task or SRB (see note below)
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit
ASC mode:	Primary, secondary or access register (AR)
Interrupt status:	Enabled or disabled for I/O and external interrupts (see note below)
Locks:	The caller may hold locks, but is not required to hold any. (See note below.)
Control parameters:	None.

Note: If the caller does not specify the EUT=YES parameter, the caller must be one of the following:

- Holding a lock
- Disabled for I/O and external interrupts
- In SRB mode.

Programming Requirements

If your program is in AR mode, issue the SYSSTATE ASCENV=AR macro before issuing SETFRR. SYSSTATE ASCENV=AR tells the system to generate code appropriate for AR mode.

For primary mode callers, the parameter list you specify on the PARMAD parameter must be in the primary address space. For AR mode callers, this parameter list can be located in any address space.

The caller must include the following mapping macros:

- IHAFRRS
- IHAPSA

SETFRR Macro

Restrictions

None.

Input Register Information

Before issuing the SETFRR macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output Register Information

When control returns to the caller, the contents of the general purpose registers (GPRs) and access registers (ARs) are unchanged, with the exception of the GPRs you specify on the WRKREGS parameter, which are used by the system.

Performance Implications

None.

Syntax

The SETFRR macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
<i>b</i>	One or more blanks must precede SETFRR.
SETFRR	
<i>b</i>	One or more blanks must follow SETFRR.

A,FRRAD= <i>FRR addr</i> R,FRRAD= <i>FRR addr</i> D P	<i>FRR addr</i> : A-type address, or register (2) - (12).
,WRKREGS=(<i>reg1,reg2</i>)	<i>reg1</i> : Decimal digits 1-15. <i>reg2</i> : Decimal digits 1-15.
,PARMAD= <i>parm area addr</i>	<i>parm area addr</i> : A-type address, or register (2) - (12). Note: This parameter may only be specified with A or R above.
,CANCEL=YES ,CANCEL=NO	Default: CANCEL=YES
,EUT=YES	
,MODE= (FULLXM PRIMARY HOME	Default: MODE=HOME

```
,
LOCAL
)
```

```
,RELATED=value
```

value: Any valid macro keyword specification.

Parameters

The parameters are explained as follows:

A,FRRAD=FRRAD *addr*

R,FRRAD=FRRAD *addr*

D

P Specifies the operation to be performed on the FRR LIFO stack:

- A An FRR address is to be added to the stack.
- R The FRR address last added to the stack is to be replaced by another FRR address.
- D The FRR address last added to the stack is to be deleted.
- P All entries in the stack are to be purged.

FRRAD specifies the address of a fullword containing the FRR address that is to be added or replaced. The parameter specifies the FRR address in a register or specifies the address of a storage location containing the FRR address.

Note: When an FRR wants to deactivate itself, **IBM recommends** that the FRR issue SETRP with REMREC=YES rather than issuing SETFRR D. See the chapter on providing recovery in *z/OS MVS Programming: Authorized Assembler Services Guide* for an explanation.

,WRKREGS=(*reg1,reg2*)

Specifies two unique general purpose registers to be used as work registers by the system.

,PARMAD=parm area *addr*

Specifies the address of a fullword to receive the address of the 24-byte parameter area initialized to zeros and provided by the system to the issuer of SETFRR. This 24-byte parameter area is in key 0 storage. If a register is specified, the address of the 24-byte parameter area is placed in the register. This parameter area is associated with the FRR address that has either been added to or has replaced an FRR address on the stack. This parameter area is passed to the FRR when an error occurs.

,CANCEL=YES

,CANCEL=NO

Specifies whether you want to allow the recovery routine to be interrupted by cancel or detach processing.

To allow a recovery routine to be interrupted, specify CANCEL=YES.

To prevent a recovery routine from being interrupted, specify CANCEL=NO. If a cancel or detach is attempted against a recovery routine for which you have specified CANCEL=NO, MVS defers cancel and detach processing until the recovery routine returns control to the system.

Usage Notes:

1. If a recovery routine that runs under the CANCEL=NO option can be called by an unauthorized program running under the same task, IBM recommends that you specify ASYNCH=NO for each ESTAE(X) macro that the recovery routine issues. This also includes any ESTAE(X) macros issued by programs that the recovery routine calls.
2. If a recovery routine running under the CANCEL=NO option calls an unauthorized program, cancel and detach processing is also deferred for the called program.

,EUT=YES

Used only with A and R, specifies that the new FRR can be used in any environment. EUT=YES is used by routines that are not certain of their environment; for example, a routine that can be called by an SRB or by a task that is executing enabled and might not hold any locks. While the FRR remains in effect, no SVCs can be issued, no new asynchronous exits are dispatched, and no vector instructions can be executed.

,MODE=options

Specifies the environment in which the FRR is to get control and also, optionally, identifies the FRRs that free critical resources. The normal or expected addressing environment is identified by FULLXM, PRIMARY, or HOME. Specify LOCAL to enable the FRR to be entered in a restricted addressing environment for freeing critical resources. Parentheses are not needed if only one option is chosen.

FULLXM

Specifies that the FRR must be entered in the same cross memory environment that existed when the SETFRR was issued.

PRIMARY

Specifies that the FRR must be entered in primary addressing mode with both the PASID and SASID the same as the PASID that existed when the SETFRR was issued, the home address space must be unchanged, and the PSW key mask must be the same as when the SETFRR was issued.

HOME

Specifies that the FRR must be entered in primary addressing mode with PASID=SASID=HASID, and the PSW key mask either the same as that at the time of the error for SRB mode, or the task storage protect key for TCB mode.

If neither FULLXM, PRIMARY, nor HOME is coded, HOME is the default.

LOCAL

Specifies that the FRR frees a critical local resource. If the FRR cannot be entered in its normal addressing environment then it must be entered in LOCAL restricted addressing environment to free resources.

For the FRR to be entered in LOCAL restricted addressing environment, a local lock must be held.

If it cannot be entered either as an FRR or as a resource manager, the FRR is skipped.

,RELATED=value

Specifies information used to self-document macros by "relating" functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user, and may be any valid coding values.

ABEND Codes

SETFRR might abnormally end with abend code X'07D'. See *z/OS MVS System Codes* for an explanation and programmer response for this code.

Return and Reason Codes

None.

Example 1

Add an FRR to the FRR stack and return the address of the parameter list to the issuer of the SETFRR. The FRR address contained in register (R5) is placed on the FRR stack in the next available FRR entry. On return, register (R2) contains the address of the parameter list associated with this FRR entry. Registers R3 and R4 are work registers used by the system.

```
SETFRR A,FRRAD=(R5),PARMAD=(R2),WRKREGS=(R3,R4)
```

Example 2

Delete the last FRR added to the FRR stack. Registers 1 and 6 are work registers used by the system.

```
SETFRR D,WRKREGS=(1,6)
```

SETFRR Macro

SETLOCK — Control Access to Serially Reusable Resources

Description

Use the SETLOCK macro to control access to serially reusable resources. Each kind of serially reusable resource is assigned a separate lock.

SETLOCK can do the following:

- Obtain a specified lock
- Release a specified lock
- Test a specified lock or determine if the lock is held on the caller's processor.

For information on using this macro on an MVS/SP version other than the current version, see "Compatibility of MVS Macros" on page 1.

Locks are discussed in the "Serialization" chapter in *z/OS MVS Programming: Authorized Assembler Services Guide*.

Note

The OBTAIN, RELEASE, and TEST options of the SETLOCK macro have the same environmental specifications, programming requirements, restrictions, register information, and performance implications described below, except where noted in the explanations of each option.

Environment

The requirements for the caller are:

Minimum authorization:	Supervisor state, PSW key 0
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any
AMODE:	24- or 31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled or disabled for I/O and external interrupts. The caller cannot be disabled when unconditionally requesting a suspend lock.
Locks:	LOCAL or CML lock must be held to obtain the CMS lock; otherwise, the caller may hold locks, but is not required to hold any. Only locks lower in the hierarchy than the lock currently being requested may be held at the time of invocation.
Control parameters:	None.

Programming Requirements

- Before you invoke the SETLOCK macro in access register mode, issue SYSSTATE ASCENV=AR.
- The caller must include the IHAPSA mapping macro.
- Before issuing an OBTAIN request for a CML lock, establish the target ASCB as either the primary or secondary address space.

Restrictions

None.

SETLOCK Macro

Input Register Information

Before issuing the SETLOCK macro, the caller must ensure that the following general purpose register (GPR) contains the specified information:

Register	Contents
13	A 5-word save area if REGS=SAVE is specified, or an 18-word save area if REGS=STDSAVE is specified.

Output Register Information

For an OBTAIN or RELEASE request, when the REGS parameter is not specified, the contents of the general purpose registers (GPRs) after control returns to the caller are as follows:

Register	Contents
0-10	Unchanged
11-12	Used as work registers by the system
13	Return code
14	Used as a work register by the system
15	Unchanged

For an OBTAIN or RELEASE request when the REGS parameter is specified, see the description of the REGS parameter for information on GPR usage.

For a TEST request, the contents of the GPRs after control returns to the caller are as follows:

Register	Contents
0-1	Unchanged.
2-12	If one of these registers is specified on the LOCKHLD parameter, that register is used as a work register by the system; otherwise, registers 2-12 are unchanged.
13-14	Unchanged.

When control returns to the caller, the access registers (ARs) contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance Implications

None.

SETLOCK OBTAIN

Syntax

The OBTAIN option of SETLOCK macro is written as follows:

name *name*: Symbol. Begin *name* in column 1.

b One or more blanks must precede SETLOCK.

SETLOCK

b One or more blanks must follow SETLOCK.

OBTAIN

,TYPE=CPU
,TYPE=CMS
,TYPE=LOCAL
,TYPE=CML,ASCB=(11)
,TYPE=CML,ASCB=*addr* *addr*: A-type address

,MODE=COND **Note:** MODE cannot be specified with TYPE=CPU.
,MODE=UNCOND

,REGS=SAVE
,REGS=USE
,REGS=STDSAVE

,RELATED=*value* *value*: Any valid macro keyword specification.

Parameters

The parameters are explained as follows:

OBTAIN
Specifies that the lock designated by the TYPE parameter is to be obtained on the caller's behalf.

,TYPE=CPU
,TYPE=CMS
,TYPE=LOCAL
,TYPE=CML,ASCB=(11)
,TYPE=CML,ASCB=*addr*
Specifies the type of lock. The types available are:

CPU The processor lock. It is a pseudo spin lock providing MVS-recognized disablement. There is one CPU lock per processor and no processor can request another processor's lock. The lock is always available. Users can obtain the CPU lock to become disabled for I/O and external interrupts.

CMS The cross memory services lock. It is a global suspend lock used to serialize functions between address spaces.

LOCAL The lock that serializes resources in the home address space pointed to by PSAAOLD. It is a local level suspend lock.

SETLOCK Macro

CML The cross memory local lock. It is a local level suspend type lock used to serialize resources in an address space other than the home address space.

The requestor of a CML lock must have authority to access the specified address space before requesting the lock. To establish authority, the requestor sets the primary or secondary address space to the one specified by the `ASCB=(11)` or `ASCB=addr` parameter. Register 11 or `addr` must contain the address of the ASCB whose local lock is requested. This address space must be nonswappable before the SETLOCK request.

Note: If the requestor specifies `OBTAIN,TYPE=CML` and the ASCB parameter points to the home address space, the request is treated as though the LOCAL lock were being obtained.

,MODE=COND

,MODE=UNCOND

Specifies whether the lock is to be conditionally or unconditionally obtained.

COND Specifies that the lock is to be conditionally obtained. That is, if the lock is not owned on another processor, it is acquired on the caller's behalf. If the lock is already held, control is returned indicating that the caller holds the lock or that another unit of work on another processor owns the lock.

UNCOND Specifies that the lock is to be unconditionally obtained. That is, if the lock is not owned on another processor, it is acquired on the caller's behalf. If the lock is already held by the caller, control is returned to the calling program indicating that it already owns the lock. If the lock is held on another processor, the system suspends the SETLOCK caller until the lock is available.

The system does not permit an unconditional OBTAIN request for a CML lock if the lock is held by a unit of work that is set nondispatchable.

,REGS=SAVE

,REGS=USE

,REGS=STDSAVE

Specifies the use of general purpose registers by the SETLOCK macro.

SAVE Specifies that the contents of registers 11 through 14 are saved in the area pointed to by register 13 and are restored upon completion of the SETLOCK request. This save area must be at least 20 bytes, and must not be the same area as the standard linkage save area used by the program.

Upon completion of the SETLOCK macro with `REGS=SAVE`, the register contents are as follows:

Register	Contents
0-14	Unchanged
15	Return code

USE Specifies that the contents of registers 11 through 13 are saved in work registers 0, 1, and 15.

Upon completion of the SETLOCK macro with REGS=USE, the register contents are as follows:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14	Used as a work register by the system
15	Return code

STDSAVE Specifies that the contents of registers 2 through 12 are saved in a standard 72-byte save area pointed to by register 13.

Upon completion of the SETLOCK macro with REGS=STDSAVE, the register contents are as follows:

Register	Contents
0-1	Unchanged
2-13	Unchanged
14	Used as a work register by the system
15	Return code

Note: See “Output Register Information” on page 32 for information on register usage when the REGS parameter is not specified.

,RELATED=value

Specifies information used to self-document macros by “relating” functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user, and may be any valid coding values.

ABEND Codes

SETLOCK might abnormally terminate with abend code X'073'. See *z/OS MVS System Codes* for an explanation and programmer response for this code.

Return and Reason Codes

When control returns to the caller, register 15 (or register 13, if the REGS parameter is not specified) contains one of the following hexadecimal return codes:

Table 4. Return Codes for the SETLOCK Macro

Return Code	Meaning and Action
00	<p>Meaning: The lock was successfully obtained.</p> <p>Action: None.</p>
04	<p>Meaning: The lock was already held by the caller.</p> <p>Action: None.</p>
08	<p>Meaning: The conditional obtain process was unsuccessful. The lock is owned by another processor.</p> <p>Action: None required. However, you might try to take some action based upon your application.</p>

When the SETLOCK OBTAIN request is for the CPU lock, the system returns only return code 0. You do not need to check the return code because once control is returned to you after the SETLOCK OBTAIN,TYPE=CPU request, you will have the CPU lock.

SETLOCK Macro

Example 1

Obtain the CPU lock, saving registers 2 through 12 in the standard save area whose address is in register 13.

```
SETLOCK  OBTAIN,TYPE=CPU,REGS=STDSAVE
```

Example 2

Obtain the CMS lock. Because the caller must hold the LOCAL or CML lock to obtain the CMS lock, the caller must first obtain the LOCAL lock unconditionally, saving registers 2 through 12 in the save area pointed to by register 13. The caller then issues the request to obtain the CMS lock.

```
SETLOCK  OBTAIN,TYPE=LOCAL,MODE=UNCOND,REGS=STDSAVE  
SETLOCK  OBTAIN,TYPE=CMS,MODE=UNCOND,REGS=STDSAVE
```

Example 3

Obtain the LOCAL lock unconditionally, saving registers 11 through 14 in the save area pointed to by register 13. The save area pointed to by register 13 is not the same area as the standard linkage save area.

```
                LR      5,13      SAVE STANDARD SAVE AREA POINTER  
                LA      13,SETLOCK_SAVEAREA  
SETLOCK  OBTAIN,TYPE=LOCAL,MODE=UNCOND,REGS=SAVE  
                LR      13,5      RESTORE STANDARD SAVE AREA POINTER  
                .  
                .  
SETLOCK_SAVEAREA DS  5F          SAVE AREA FOR SETLOCK REQUESTS
```

SETLOCK RELEASE

Syntax

The RELEASE option of the SETLOCK macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede SETLOCK.
SETLOCK	
b	One or more blanks must follow SETLOCK.

RELEASE

```
,TYPE=CPU  
,TYPE=CMS  
,TYPE=LOCAL  
,TYPE=CML,ASCB=(11)  
,TYPE=CML,ASCB=addr          addr: A-type address  
  
,REGS=SAVE  
,REGS=USE
```

,REGS=STDSAVE

,RELATED=*value*

value: Any valid macro keyword specification.

Parameters

The parameters are explained as follows:

RELEASE

Specifies that the lock designated by the TYPE parameter is to be released.

Note: If you specify RELEASE,TYPE=CML,ASCB=(11) or ASCB=*addr*, the ASCB parameter specifies the home address space, and the lock that the caller holds is home's local lock, then SETLOCK processing treats the CML release request as a RELEASE, TYPE=LOCAL.

,TYPE=CPU

,TYPE=CMS

,TYPE=LOCAL

,TYPE=CML,ASCB=(11)

,TYPE=CML,ASCB=*addr*

Specifies the type of lock. The types available are:

CPU

The processor lock. It is a pseudo spin lock providing MVS-recognized disablement. There is one CPU lock per processor and no processor can request another processor's lock. The lock is always available. Users can obtain the CPU lock to become disabled for I/O and external interrupts.

CMS

The cross memory services lock. It is a global suspend lock used to serialize functions between address spaces.

LOCAL

The lock that serializes resources in the home address space pointed to by PSAAOLD. It is a local level suspend lock.

CML

The cross memory local lock. It is a local level suspend type lock used to serialize resources in an address space other than the home address space.

The requestor of a CML lock must have authority to access the specified address space before requesting the lock. To establish authority, the requestor sets the primary or secondary address space to the one specified by the ASCB=(11) or ASCB=*addr* parameter. Register 11 or *addr* must contain the address of the ASCB whose local lock is requested. This address space must be nonswappable before the SETLOCK request.

Note: If the requestor specifies OBTAIN,TYPE=CML and the ASCB parameter points to the home address space, the request is treated as though the LOCAL lock were being obtained.

,REGS=SAVE

,REGS=USE

,REGS=STDSAVE

Specifies the use of general purpose registers by the SETLOCK macro.

SETLOCK Macro

SAVE Specifies that the contents of registers 11 through 14 are saved in the area pointed to by register 13 and are restored upon completion of the SETLOCK request. This save area must be at least 20 bytes, and must not be the same area as the standard linkage save area used by the program.

Upon completion of the SETLOCK macro with REGS=SAVE, the register contents are as follows:

Register	Contents
0-14	Unchanged
15	Return code

USE Specifies that the contents of registers 11 through 13 are saved in work registers 0, 1, and 15.

Upon completion of the SETLOCK macro with REGS=USE, the register contents are as follows:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14	Used as a work register by the system
15	Return code

STDSAVE Specifies that the contents of registers 2 through 12 are saved in a standard 72-byte save area pointed to by register 13.

Upon completion of the SETLOCK macro with REGS=STDSAVE, the register contents are as follows:

Register	Contents
0-1	Unchanged
2-13	Unchanged
14	Used as a work register by the system
15	Return code

Note: See “Output Register Information” on page 32 for information on register usage when the REGS parameter is not specified.

,RELATED=value

Specifies information used to self-document macros by “relating” functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user, and may be any valid coding values.

Return and Reason Codes

When control is returned, register 15 (or register 13, if the REGS parameter is not specified) contains one of the following hexadecimal return codes:

Table 5. Return Codes for SETLOCK RELEASE

Return Code	Meaning and Action
00	Meaning: The lock was successfully released. Action: None.
04	Meaning: The lock was not owned. The lock was free when the release request was issued. Action: None.

Table 5. Return Codes for SETLOCK RELEASE (continued)

Return Code	Meaning and Action
08	Meaning: The release process was unsuccessful. The lock was owned by a different processor. Action: None required. However, you might try to take some action based upon your application.
0C	Meaning: The release process was unsuccessful. The caller does not own the specified local or CML lock. This return code applies to LOCAL or CML release only. Action: None required. However, you might try to take some action based upon your application.

Example 1

Release the local lock and check the return code from the SETLOCK request. If the release was unsuccessful, branch to the code at the RLSEFAIL label.

```
SETLOCK  RELEASE,TYPE=LOCAL
LTR      13,13
BNZ      RLSEFAIL
```

Example 2

Release the CML lock, saving the contents of registers 2 through 12 in a standard save area. Check the return code from the SETLOCK request, and branch to the code at the RLSEFAIL label if the release was unsuccessful.

```
SETLOCK  RELEASE,TYPE=CML,REGS=STDSAVE
LTR      15,15
BNZ      RLSEFAIL
```

SETLOCK TEST

Syntax

The TEST option of the SETLOCK macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
<i>b</i>	One or more blanks must precede SETLOCK.
SETLOCK	
<i>b</i>	One or more blanks must follow SETLOCK.
TEST	
,TYPE=CPU	
,TYPE=CMS	
,TYPE=LOCAL	
,TYPE=ALOCAL	
,TYPE=CML	Note: LOCKHLD or ASCB must be specified with TYPE=CML.

SETLOCK Macro

<code>,LOCKHLD=(reg)</code>	<i>reg</i> : Register (2) - (12) Note: LOCKHLD is valid only with TYPE=CML, TYPE=ALOCAL, and TYPE=CPU
<code>,ASCB=(reg)</code>	<i>reg</i> : Register (2) - (12) Note: ASCB is valid only with TYPE=CML.
<code>,BRANCH=(HELD,addr)</code> <code>,BRANCH=(NOTHELD,addr)</code>	<i>addr</i> : RX-type address.
<code>,RELATED=value</code>	<i>value</i> : Any valid macro keyword specification.

Parameters

The parameters are explained as follows:

TEST

Specifies that the lock designated by the TYPE parameter is to be checked to determine if it is currently held on the requesting processor.

,TYPE=CPU

,TYPE=CMS

,TYPE=LOCAL

,TYPE=ALOCAL

,TYPE=CML

Specifies the type of lock. The types available are:

CPU

The processor lock. It is a pseudo spin lock providing MVS-recognized disablement. There is one CPU lock per processor, and no processor can request another processor's lock. The lock is always available. Users can obtain the CPU lock to become disabled for I/O and external interrupts.

CMS

The cross memory services lock. It is a global suspend lock used to serialize functions between address spaces.

LOCAL

The lock that serializes resources in the home address space pointed to by PSAAOLD. It is a local level suspend lock.

ALOCAL

Determines whether a local lock is held, either home's LOCAL lock or a CML lock. The LOCKHELD=(*reg*) parameter can be specified with TYPE=ALOCAL.

CML

The cross memory local lock. It is a local level suspend type lock used to serialize resources in an address space other than the home address space. TYPE=CML specifies that the caller wishes to determine whether a CML lock is held. Either the ASCB=(*reg*) or the LOCKHLD=(*reg*) parameter can be specified with TYPE=CML, but not both.

,LOCKHLD=(reg)

Specifies that the designated register is to be used as a return register by the macro. This parameter is valid only for TYPE=CML, TYPE=CPU, and TYPE=ALOCAL.

SETLOCK Macro

If TYPE=CML is specified and a CML lock is held, the system returns the ASCB address of the CML-locked address space in the specified register.

If TYPE=CPU is specified, the system returns the current CPU lock use count for this processor in the specified register.

If TYPE=ALOCAL is specified and the LOCAL lock is held, the system returns a zero in the specified register.

If TYPE=ALOCAL is specified and a CML lock is held, the system returns the ASCB address of the CML-locked address space in the specified register. If a local lock is held, the system returns a zero in the specified register.

,ASCB=(reg)

Specifies a register that contains the ASCB address. The system checks the ASCB to determine whether the requestor's local lock is a CML lock. This parameter is valid only with TYPE=CML.

Note: Unlike the OBTAIN and RELEASE options of the SETLOCK macro, ASCB=addr is not valid.

,BRANCH=(HELD,addr)

,BRANCH=(NOTHELD,addr)

If (HELD,addr) is specified, the specified address is branched to if the specified lock is held on the requesting processor.

If (NOTHELD,addr) is specified, the specified address is branched to if the specified lock is not currently held on the requesting processor.

,RELATED=value

Specifies information used to self-document macros by “relating” functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user, and may be any valid coding values.

Return and Reason Codes

When control is returned, register 15 contains one of the following hexadecimal return codes (if the BRANCH= parameter was omitted):

Table 6. Return Codes for SETLOCK TEST

Return Code	Meaning and Action
00	Meaning: The lock was held by the requestor, or (if TYPE=CMS was specified) at least one lock was held. Action: None.
04	Meaning: The lock was available, or (if TYPE=CMS was specified) no lock was held. Action: None.

Note: TYPE=CMS is used to determine if at least one cross memory services lock is held, but cannot be used to determine which one, or to determine if all are held.

Example 1

If a local lock is not held, branch to DSRLINT; otherwise, execute the next sequential instruction.

```
SETLOCK TEST,TYPE=LOCAL,BRANCH=(NOTHELD,DSRLINT)
```

SETLOCK Macro

Example 2

Put the current CPU lock use count for this processor into register 3.

```
SETLOCK TEST,TYPE=CPU,LOCKHLD=(3)
```

Example 3

Determine whether the local lock of the address space specified in register 11 is held as a CML lock.

```
SETLOCK TEST,TYPE=CML,ASCB=(11)
```

SETRP — Set Return Parameters

Description

Use the SETRP macro within a recovery routine to indicate the various requests that the recovery routine can make. SETRP is valid for functional recovery routines (FRRs) and ESTAE-type recovery routines. For more information about recovery routines, see the section on providing recovery in *z/OS MVS Programming: Authorized Assembler Services Guide*.

The SETRP macro is also described in *z/OS MVS Programming: Assembler Services Reference ABE-HSP* with the exception of the RECORD, FRELOCK, SERIAL, RETRY, RETRY15, FRLKRTY, and SSRESET parameters.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state and any PSW key. For the RECORD, FRELOCK, SERIAL, RETRY, RETRY15, AND FRLKRTY parameters, supervisor state or PSW key 0-7.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31- or 64-bit Note: For the DUMPOPX parameter, AMODE=64 is not allowed.
ASC mode:	Primary, secondary, or access register (AR) Note: Callers in secondary ASC mode cannot specify the DUMPOPX keyword.
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	The caller may hold locks, but is not required to hold any.
Control parameters:	None.

Programming Requirements

- If the program is in AR mode, issue the SYSSTATE ASCENV=AR macro before SETRP. SYSSTATE ASCENV=AR tells the system to generate code appropriate for AR mode.
- Include the IHASDWA mapping macro to map the system diagnostic work area (SDWA). The SDWA is addressable when the recovery routine is entered; when the SETRP macro is issued, the same address space must be addressable. (See SDWA in *z/OS MVS Data Areas, Vol 4 (RD-SRRA)* for the mapping provided by IHASDWA.)
- If you plan to specify RETREGS=YES,RUB=*reg info addr*, you must obtain storage for and initialize the register update block (RUB). See the RETREGS parameter description for more information about this area.

Restrictions

- You can use SETRP only if the system provided an SDWA.
- Recovery routines established through the STAE macro, or the STAI parameter on the ATTACH or ATTACHX macro, cannot update registers on retry, so the RETREGS parameter does not apply.
- For FRRs, RETREGS=YES (or RETREGS=NO) has no effect. For FRRs, the system always restores GPRs 0-14 from the SDWASRSV field, and ARs 0-14

SETRP Macro

from the SDWAARSV field. If you specify RETRY15=YES, the system also restores GPR 15 and AR 15 from the SDWASRSV and SDWAARSV fields, respectively.

Input Register Information

Before issuing the SETRP macro, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register	Contents
1	If you do not specify the WKAREA parameter, address of the SDWA; otherwise, the caller does not have to place any information into this register.
13	If you specify the REGS parameter, address of a standard 72-byte save area containing the registers to be restored; otherwise, the caller does not have to place any information into this register.

Before issuing the SETRP macro, the caller must ensure that the following access registers (ARs) contain the specified information:

Register	Contents
1	If you do not specify the WKAREA parameter, ALET of the SDWA whose address is in GPR 1; otherwise, the caller does not have to place any information into this register.
13	If you specify the REGS parameter, ALET of the standard 72-byte save area whose address is in GPR 13; otherwise, the caller does not have to place any information into this register.

Output Register Information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

When control returns to the caller, the access registers (ARs) contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

Note: Control does not return to the caller if the caller specifies the REGS parameter.

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance Implications

None.

Syntax

The SETRP macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
	One or more blanks must precede SETRP.
SETRP	
	One or more blanks must follow SETRP.
WKAREA=(<i>reg</i>)	<i>reg</i> : Decimal digits 1-12. Default: WKAREA=(1)
,REGS=(<i>reg1</i>) ,REGS=(<i>reg1,reg2</i>)	<i>reg1</i> : Decimal digits 0-12, 14, 15. <i>reg2</i> : Decimal digits 0-12, 14, 15. Note: If you specify (<i>reg1,reg2</i>), specify the registers in the same order as in an STM instruction; for example, to restore all registers except register 13, specify REGS=(14,12).
,DUMP=IGNORE ,DUMP=YES ,DUMP=NO	Default: DUMP=IGNORE
,DUMPOPT= <i>parm list addr</i> ,DUMPOPX= <i>parm list addr</i>	<i>parm list addr</i> : RX-type address, or register (2) - (12). Note: Specify these parameters only if you specify DUMP=YES.
,RC=0 ,RC=4 ,RC=16	Default: RC=0
,RETADDR= <i>retry addr</i>	<i>retry addr</i> : RX-type address, or register (2) - (12). Note: This parameter may be specified only if RC=4 is specified above.
,RETREGS=NO ,RETREGS=YES ,RETREGS=YES,RUB= <i>info addr</i> ,RETREGS=64	<i>info addr</i> : RX-type address, or register (2) - (12). Default: RETREGS=NO Note: This parameter may be specified only if RC=4 is specified above.
,FRESDDWA=NO ,FRESDDWA=YES	Default: FRESDDWA=NO Note: This parameter may be specified only if RC=4 is specified above.
,COMPCOD= <i>code1</i> ,COMPCOD=(<i>code</i>) ,COMPCOD=(<i>code</i> , USER) ,COMPCOD=(<i>code</i> ,SYSTEM)	<i>code1</i> : Symbol or decimal number. <i>code</i> : Symbol, decimal number, or register (2) - (12). Default: COMPCOD=(<i>code</i> ,USER)
,FRELOCK=(<i>locks</i>)	<i>locks</i> : Any combination of the following, separated by commas: CPU CMS LOCAL CML(<i>cmlascb</i>) <i>cmlascb</i> : RX-type address or register (2) - (12).

SETRP Macro

,REASON= <i>code</i>	<i>code</i> : Symbol, decimal or hexadecimal number, or register (2) - (12).
,RECORD=IGNORE ,RECORD=YES ,RECORD=NO	Default: RECORD=IGNORE
,RECPARM= <i>record list addr</i>	<i>record list addr</i> : RX=type address, or register (2) - (12). Note: This parameter may be specified only if RECORD=IGNORE or RECORD=YES is specified above.
,SERIAL=YES ,SERIAL=NO	
,RETRY=FRR ,RETRY=ERROR	Default: RETRY=FRR
,RETRY15=NO ,RETRY15=YES	Default: RETRY15=NO
,REMREC=NO ,REMREC=YES	Default: REMREC=NO
,FRLKRTY=NO ,FRLKRTY=YES	Default: FRLKRTY=NO
,SSRESET=YES ,SSRESET=NO	

Parameters

The parameters are explained below:

WKAREA=(*reg*)

Specifies the address of the SDWA passed to the recovery routine. If this parameter is omitted, the address of the SDWA must be in register 1.

,REGS=(*reg 1*)

,REGS=(*reg 2*)

Specifies the register or range of registers to be restored from the 72-byte standard save area pointed to by the address in register 13. If you specify REGS, a branch on register 14 instruction will also be generated to return control to the system. If you do not specify REGS, you must code your own branch on whichever register contains the return address.

Note: If you specify *reg1,reg2*, specify the registers in the same order as in an STM instruction; for example, to restore all registers except register 13, specify REGS=(14,12).

,DUMP=IGNORE

,DUMP=YES

,DUMP=NO

Specifies that the dump option fields will not be changed (IGNORE), will be zeroed (NO), or will be merged with dump options specified in previous dump requests, if any (YES). If IGNORE is specified, a previous recovery routine had

requested a dump or a dump had been requested through the ABEND macro, and the previous request will remain intact. If NO is specified, no dump will be taken.

DUMP=YES does not guarantee that a SYSABEND/SYSUDUMP will be taken. You may specify this request in an FRR for an SRB but you will get an abdump only if the SRB abend successfully percolates to a task *and* none of the FRRs for that task choose to retry *and* the final value of the DUMP= remains the same after every recovery routine has received control.

,DUMPOPT=*parm list addr*

,DUMPOPX=*parm list addr*

Specifies the address of a parameter list of dump options. You can create the parameter list through the list form of the SNAP or SNAPX macro, or you can create a compatible list. DUMPOPT specifies the address of a parameter list that the SNAP macro creates. DUMPOPX specifies the address of a parameter list that the SNAPX macro creates. A program in secondary mode cannot use the DUMPOPX parameter.

If the specified dump options include subpools for storage areas to be dumped, up to seven subpools can be dumped. Subpool areas are accumulated and wrapped, so that the eighth subpool area specified replaces the first.

If the dump options specified include ranges of storage areas to be dumped, only the storage areas in the first thirty ranges will be dumped.

The TCB, DCB, ID, and STRHDR options available on SNAP or SNAPX are ignored if they appear in the parameter list. The TCB used will be the one for the task that encountered the error. The DCB used will be one created by the system, and either SYSABEND, SYSMDUMP, or SYSUDUMP will be used as a DDNAME.

,REASON=*code*

Specifies the reason code that the user wishes to pass to subsequent recovery routines. The value range for *code* is any 32-bit hexadecimal number or 31-bit decimal number. See *z/OS MVS Programming: Assembler Services Reference ABE-HSP* for information about how a user can change this code.

,RC=0

,RC=4

,RC=16

Specifies the return code the recovery routine sends to the system to indicate what further action is required:

Decimal Code Meaning

0	Continue with error processing; causes entry into previously-specified recovery routine, if any.
4	Retry using the retry address specified.
16	Valid only for an ESTAI/STAI recovery routine. The system should not give control to any further ESTAI/STAI routines, and should abnormally end the task.

,RETADDR=*retry addr*

Specifies the address of the retry routine to which control is to be given.

,RETREGS=NO

,RETREGS=YES

,RETREGS=YES,RUB=*reg info addr*

SETRP Macro

,RETREGS=64

Specifies the contents of the registers to be restored on entry to the retry routine. RETREGS=NO indicates that you do not want the system to restore any register contents from the SDWA.

If you specify RETREGS=YES, in a recovery routine defined through the ESTAE, ESTAEX, or FESTAE macro, the ESTAI parameter on the ATTACH or ATTACHX macro, or an associated recovery routine (ARR), the system does the following:

- Initializes GPRs 0-15 from the SDWASRSV field of the SDWA
- Initializes ARs 0-15 from the SDWAARSV field of the SDWA.

Specifying RETREGS=64 is the same as specifying RETREGS=YES, except the registers for retry are the 64-bit general purpose registers in field SDWAG64.

RUB (register update block) specifies the address of an area that contains register update information for the GPRs. The data you specify in this area will be moved into the SDWASRSV field of the SDWA and will be loaded into the GPRs on entry to the retry routine. You cannot use the RUB to specify data to be moved into the SDWAARSV field for loading the ARs. You can use the RUB for both ESTAE-type recovery routines and FRRs. The maximum length of the RUB is 66 bytes. You must acquire storage for and initialize this area as follows:

- The first two bytes represent the registers to be updated, register 0 corresponding to bit 0, register 1 corresponding to bit 1, and so on. The user indicates which of the registers are to be stored in the SDWA by setting the corresponding bits in these two bytes.
- The remaining 64 bytes contain the update information for the registers, in the order 0-15. If all 16 registers are being updated, this field consists of 64 bytes. If only one register is being updated, this field consists of only 4 bytes for that one register.

For example, if only registers 4, 6, and 9 are being updated:

- Bits 4, 6, and 9 of the first two bytes are set.
- The remaining field consists of 12 bytes for registers 4, 6, and 9; the first 4 bytes are for register 4, followed by 4 bytes for register 6, and 4 final bytes for register 9.

,FRESDDWA=NO

,FRESDDWA=YES

Specifies that the entire SDWA be freed (YES) or not be freed (NO) before entry into the retry routine.

,COMPCOD=*code*1

,COMPCOD=(*code*)

,COMPCOD=(*code*,USER)

,COMPCOD=(*code*,SYSTEM)

Specifies the user or system completion code that the user wants to pass to subsequent recovery routines.

,FRELOCK=(*locks*)

Specifies the locks to be freed and the corresponding lockwords that are placed in the SDWA:

CPU

Processor lock

CMS

Cross memory services lock

LOCAL	Storage lock of the storage the caller is executing in
CML(<i>cmlascb</i>)	Cross memory local lock, where <i>cmlascb</i> indicates the ASCB address of the address space for which the local lock is to be freed

Note: If FRLKRTY=NO is specified or taken as a default, the specified locks are freed only on percolation, not on retry. Specifying FRLKRTY=YES allows the locks listed in FRELOCK to be freed on retry.

,RECORD=IGNORE

,RECORD=YES

,RECORD=NO

Specifies whether the SDWA is to be recorded in SYS1.LOGREC (RECORD=YES/NO), or whether the system should honor previous instructions about recording the SDWA in SYS1.LOGREC (RECORD=IGNORE).

If you specify RECORD=YES, the system records the entire SDWA (including the fixed length base, the variable length recording area, and the recordable extensions) in SYS1.LOGREC when the ESTAE recovery routine returns control, even if the mainline program issued the ESTAE or ESTAEX macro with RECORD=NO.

If you specify RECORD=NO, the system does not record the SDWA in SYS1.LOGREC, even if the mainline program issued ESTAE or ESTAEX with RECORD=YES.

If you specify RECORD=IGNORE, the system honors the request as specified by the RECORD parameter on the ESTAE or ESTAEX macro.

,RECPARM=record list addr

Specifies the address of a user-supplied record parameter list used to update the SDWA with recording information. The parameter list consists of three 8-byte fields:

- The first field contains the load module name.
- The second field contains the CSECT name (assembly module name).
- The third field contains the recovery routine name (assembly module name). If the recovery routine label is not the same as the assembly module name, the label can be used.

The three fields are left-justified, and padded with blanks.

,SERIAL=YES

,SERIAL=NO

Specifies whether the percolation from an SRB mode FRR to a related task recovery routine (ESTAE or FRR) is to be serialized (YES) or not serialized (NO) with respect to unlocked task recovery. See 'SRB to Task Percolation' in *z/OS MVS Programming: Authorized Assembler Services Guide*.

If the task is already in recovery for another error when SERIAL=YES is specified, the percolation request is deferred pending a requested task retry from any recovery routine covering mainline code. If such a retry is not requested, the task is terminated and all deferred percolations are purged. Only the last FRR to receive control when an error occurs can specify SERIAL=YES.

,RETRY=FRR

,RETRY=ERROR

Specifies the cross memory environment in which the retry routine gets control.

SETRP Macro

RETRY=FRR, the default, specifies that the retry routine gets control in the cross memory environment that exists at the time of entry to the FRR.

RETRY=ERROR specifies that the retry routine gets control in the cross memory environment that existed at the time of the error. Do not specify RETRY=ERROR if the cross memory status at the time of the error is not available, that is, if SDWARPIV is set to one. (Be careful not to create a loop by retrying to an erroneous cross memory state with RETRY=ERROR.)

,RETRY15=YES

,RETRY15=NO

In an FRR environment only, specifies that GPR 15 is restored from SDWASRSV and AR 15 is restored from SDWAARSV if RETRY15=YES. Otherwise, it contains the entry point address of the retry routine.

This parameter may be specified only when RC=4 is specified. If RETRY15=YES is not coded on any SETRP invocation prior to returning to the system, the effect is that of specifying RETRY15=NO.

,REMREC=YES

,REMREC=NO

In an FRR or ESTAE environment, specifies that the FRR/ESTAE entry for the currently running FRR/ESTAE routine be removed (REMREC=YES) or not removed (REMREC=NO). This parameter may be specified only when RC=4 is specified, indicating a retry request.

The entry is removed before control returns to the retry point. If REMREC=YES is not coded on any SETRP invocation before the system receives control, the effect is that of specifying REMREC=NO. The REMREC parameter may be used to remove a recovery routine that has been defined with a token, although the token cannot be specified when you code the SETRP macro.

,FRLKRTY=YES

,FRLKRTY=NO

In an FRR environment only, specifies that the locks specified on FRELOCK be freed (FRLKRTY=YES) or not be freed (FRLKRTY=NO) on retry.

This parameter may be specified only when RC=4 is specified. If FRLKRTY=YES is not coded on any SETRP invocation prior to returning to the system, the effect is that of specifying FRLKRTY=NO.

SSRESET=YES

SSRESET=NO

SSRESET=YES specifies that, if the current recovery routine abnormally ends, the next recovery routine is to get control in the subspace environment that existed when the current recovery routine was entered. Specify SSRESET=YES when the current recovery routine has temporarily modified the subspace environment, and when it is appropriate for the next recovery routine to receive control in the subspace environment in which the current recovery routine received control.

SSRESET=NO negates an earlier specification of SSRESET=YES. Specify SSRESET=NO when SSRESET=YES protection is no longer needed. If the current recovery routine abnormally ends after specifying SSRESET=NO, the next recovery routine will get control in the subspace in which the current routine was running when the error occurred.

If you do not specify SSRESET and the current recovery routine abnormally ends, the next recovery routine will get control in the subspace in which the current recovery routine was running when the error occurred.

See the chapter on subspaces in *z/OS MVS Programming: Extended Addressability Guide* for more information about subspaces and recovery.

Notes:

1. The FRESDDWA parameter cannot be specified or defaulted for a functional recovery routine (FRR). The SDWA is always released before an FRR's retry routine gets control.
2. The SERIAL parameter is relevant only for FRRs defined for SRBs that have a related task.
3. The SERIAL and RETRY parameters are mutually exclusive.

The following table indicates which parameters are available to functional recovery routines (FRRs) and which parameters are available to ESTAE-type recovery routines.

Parameter	FRR	ESTAE-type recovery routines
WKAREA	x	x
REGS	x	x
DUMP	x	x
REASON	x	x
RC=0	x	x
RC=4	x	x
RC=16		x
RETADDR	x	x
RETREGS	x	x
RUB	x	x
FRESDDWA		x
COMPCOD	x	x
FRELOCK	x	
RECORD	x	x
RECPARM	x	x
SERIAL	x	
RETRY	x	
RETRY15	x	
REMREC	x	x
FRLKRTY	x	
DUMPOPT	x	x
DUMPOPX	x	x
SSRESET		x

ABEND Codes

None.

Return and Reason Codes

None.

Example 1

The first FRR established for an SRB routine requests percolation, freeing of the CML lock (the ASCB address is in register 2), and serialization of percolation to the related task.

```
SETRP RC=0,FRELOCK=(CML(2)),SERIAL=YES
```

SETRP Macro

Example 2

An FRR requests retry with the retry routine getting control in the same cross memory mode as the time of FRR entry. The retry address is in register 3.

```
SETRP RC=4,RETADDR=(3),RETRY=FRR
```

SJFREQ — Call Scheduler JCL Facility Services

Description

The SJFREQ macro services can be used to manipulate text unit data that represents processing options for system output (sysout) data sets. The SJFREQ services described in this section include the following:

- The SJFREQ RETRIEVE service retrieves keyword subparameter information in text unit format from output descriptors. These output descriptors can be specified either on an OUTPUT JCL statement or through dynamic output.
- The SJFREQ SWBTU_MERGE service merges lists of scheduler work block text units (SWBTUs) and allows applications to indicate keys to be removed from a list of SWBTUs.
- The SJFREQ VERIFY service verifies OUTDES statements, operands, and subparameters and builds text units to represent them. Your application can use these text units to dynamically define processing options for a sysout data set.
- The SJFREQ TERMINATE service cleans up SJF's recovery and working storage environment.

z/OS MVS Programming: Authorized Assembler Services Guide describes the OUTDES statement and its operands, as well as the individual SJF services.

Environment

The requirements for the caller are:

Minimum authorization:	The requirements vary, depending on the service. <ul style="list-style-type: none">• SJFREQ RETRIEVE and SJFREQ SWBTU_MERGE: Supervisor state, with a PSW key that matches the key of the caller's storage.• SJFREQ VERIFY: Problem state or supervisor state. For supervisor state, the caller must run in PSW key 1 and the caller's storage must be in PSW key 1. For problem state, the caller must have a PSW key that matches the key of the caller's storage.• SJFREQ TERMINATE: Problem state or supervisor state, with a PSW key that matches the key of the caller's storage.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	Any
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in the primary address space

Programming Requirements

The application must include the CVT and IEFJESCT mapping macros. If the application codes the PARM parameter on the SJFREQ macro, the caller must also declare a 4-byte pointer, SJFPTR.

SJFREQ Macro

For each SJFREQ invocation, the application must initialize certain fields in an input parameter list. Fields are discussed within each service description. The following lists the parameter list name for each service.

Service	Parameter List Name
RETRIEVE	IEFSJREP
SWBTU_MERGE	IEFSJSMP
VERIFY	IEFSJVEP
TERMINATE	Any of the three parameter list names above.

Restrictions

None.

Input Register Information

Before issuing the SJFREQ macro, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register	Contents
1	If PARM is not specified, the address of a word that contains the address of the input parameter list
13	The address of an 18-word save area

Output Register Information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14	Used as a work register by the system
15	Return code

When control returns to the caller, the ARs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance Implications

None.

Syntax

The standard form of the SJFREQ macro is written as follows:

name

name: Symbol. Begin *name* in column 1.

b One or more blanks must precede SJFREQ

SJFREQ

b One or more blanks must follow SJFREQ

REQUEST=RETRIEVE
 REQUEST=SWBTU_MERGE
 REQUEST=VERIFY
 REQUEST=TERMINATE

,PARM=*addr* *addr*. RX-type address, or registers (1) - (12).

Parameters

The parameters are explained as follows:

REQUEST=RETRIEVE
REQUEST=SWBTU_MERGE
REQUEST=VERIFY
REQUEST=TERMINATE

Specifies the SJF service to be called. SJF services that you can request through the SJFREQ macro are:

RETRIEVE

The SJFREQ RETRIEVE service retrieves keyword subparameter information in text unit format from output descriptors. These output descriptors can be specified either on an OUTPUT JCL statement or through dynamic output. See “SJFREQ RETRIEVE Service” on page 56 for more detailed information on using this service.

SWBTU_MERGE

Use SJFREQ SWBTU_MERGE to create a single list of SWBTUs from a base SWBTU list and a merge SWBTU list. The resulting list of SWBTUs contains all the text units in the base and merge lists. If duplicate text units exist, only one appears in the final SWBTU list. The SWBTU_MERGE service also allows an application to indicate keys to be removed from a list of SWBTUs. See “SJFREQ SWBTU_MERGE Service” on page 60 for more detailed information on using this service.

VERIFY

Use SJFREQ VERIFY to validate OUTDES statements and build text units to be used as input to dynamic output. See “SJFREQ VERIFY Service” on page 69 for more detailed information on using this service.

TERMINATE

Use SJFREQ TERMINATE after previously issuing a RETRIEVE, SWBTU_MERGE, or VERIFY request that specified the *no cleanup* option (SJxxNOCU). TERMINATE cleans up SJF's recovery and working storage environment. See "SJFREQ TERMINATE Service" on page 82 for more detailed information on using this service.

,**PARM**=*addr*

Specifies the address of the parameter list for the service requested. You must initialize certain parameter list fields for each service. The following list indicates where the parameter lists are described.

- "SJFREQ RETRIEVE Input Parameters" on page 57 describes the RETRIEVE parameter list.
- "SJFREQ SWBTU_MERGE Input Parameters" on page 60 describes the SWBTU_MERGE parameter list.
- "SJFREQ VERIFY Input Parameters" on page 69 describes the VERIFY parameter list.
- You may only use IEFSJREP, IEFSJSMP, or IEFSJVEP (use the same one you used on a prior request) for the TERMINATE service.

If you omit **PARM**, register 1 must contain the address of a word that contains the address of the input parameter list.

SJFREQ RETRIEVE Service

The SJFREQ RETRIEVE service retrieves keyword subparameter information in text unit format from output descriptors. These output descriptors can be specified either on an OUTPUT JCL statement or through dynamic output. Your application can invoke this service to retrieve output descriptor information in a functional subsystem environment.

Programming Requirements

An application must complete the following steps to issue a RETRIEVE request:

1. Obtain storage for the RETRIEVE parameter list (IEFSJREP) and initialize the fields. Place the address of IEFSJREP in the PARM parameter.
2. Provide a keyword list (SJRELIST). The keyword list contains the keywords for which the application wants information retrieved. SJFREQ RETRIEVE Keyword List describes the list.
3. Issue SJFREQ RETRIEVE.

SJFREQ RETRIEVE Keyword List

The keyword list contains paired fields; each pair consists of a keyword field and a pointer field. In the list, the application specifies the JCL keywords for which information is to be retrieved. For each keyword specified, the RETRIEVE service returns in SJRETPAD a pointer to the text unit pointer list associated with the keyword.

The following table shows the SJRELIST paired fields and their offsets and lengths. The fields that the application initializes are indicated.

Field Name	Offset (bytes)	Value Length (bytes)	Value to be assigned
SJRELIST			
SJREKEYW	0 (X'0')	8	keyword 1 (initialized by application)
SJRETPAD	8 (X'8')	4	pointer (returned by the system)
SJREKEYW	12 (X'C')	8	keyword 2 (initialized by application)
SJRETPAD	20 (X'14')	4	pointer (returned by the system)
.	.	.	.
.	.	.	.

In addition to JCL keywords, SJFREQ can also return a text unit for the keyword providing special TCP/IP support. This text unit has the following characteristics in the Output Descriptor.

Keyword Parameter: IPADDR
Key in Hex: 8005
Maximum number of Value Fields: 1
Length of Value Field: 0 - 124
Value Field: EBCDIC text '40'X - 'FE'X
Function: IP address extracted from the DEST='IP:ip-address' format of the DEST=keyword

SJFREQ RETRIEVE Input Parameters

In addition to providing a keyword list, for each SJFREQ invocation, you need to initialize certain fields of parameter list IEFSJREP. The list of parameters and descriptions of their values are below.

SJREID The identifier 'SJRE' of the RETRIEVE parameter list. Assign the symbolic equate SJRECID to this field.

SJREVERS The current version number of the RETRIEVE parameter list. Assign the symbolic equate SJRECVVER to this field.

SJREFLAG The environment control flag.

SJRENOCU Indicates whether the SJF environment is preserved from call to call or obtained for each call. Set to one to preserve the environment from call to call. Set to zero to obtain a new environment for a call.

SJRELEN The length of the RETRIEVE parameter list (IEFSJREP). Assign the symbolic equate SJRELGTH to this field.

SJRESTOR The local working storage pointer or zero. This field must contain zero on the first RETRIEVE call. On subsequent calls, the field contains the value returned from the most recent RETRIEVE call. Use the returned value for subsequent calls.

SJREJDVT Enter zero for this field on input. This field must contain zero on the first RETRIEVE call. On subsequent calls, the field contains the value returned from the most recent RETRIEVE call. Use the returned value for subsequent calls.

SJRETOKN Enter the output descriptor token in this field. If you are coding a

SJFREQ Macro

Print Service Facility (PSF) exit, refer to *PSF/MVS System Programming Guide* for information on obtaining the output descriptor token. The token can be obtained from the GDSOUTK field in data area IAZFSIP.

- SJREAREA** The address of the storage area in which RETRIEVE is to return the keyword subparameter information in text unit format from output descriptors.
- SJRESIZE** The amount of the storage allocated for the output descriptor information.
- SJRENKWD** The number of keywords passed in the keyword list (SJRELIST).
- SJREKWDL** The address of the keyword list (SJRELIST).

SJFREQ RETRIEVE Output Parameters

In addition to the output provided in the keyword list, the RETRIEVE service returns data in several fields of the IEFSJREP parameter list. The list of output parameters and their descriptions are below.

- SJREREAS** Contains a reason code returned from the RETRIEVE service. This field contains a value when register 15 contains a return code other than zero.
- SJREKERR** Contains the address of the first keyword in the keyword list that caused the error.
- SJREPAD** Contains the address of the list of text unit pointers. Each text unit pointer points to a returned text unit. The last text unit has the high-order bit set to one.

ABEND Codes

None.

SJFREQ RETRIEVE Return and Reason Codes

SJFREQ RETRIEVE return codes appear in register 15. When SJFREQ returns control to your program, SJREREAS contains a reason code when register 15 contains a nonzero value. Return and reason codes are defined in macro IEFSJRC. The following table identifies the hexadecimal return and reason code combinations, tells what each means, and recommends an action you should take.

Table 7. Return and Reason Codes for the SJFREQ RETRIEVE Service

Return Code	Reason Code	Meaning and Action
0	000	Meaning: RETRIEVE processing completed successfully. Action: None.
4	002	Meaning: Program error. The token specified in SJRETOKN is not valid. Action: Specify a valid token in SJRETOKN.
4	004	Meaning: Program error. The RETRIEVE request was not processed. The value specified in SJREJDVT is not valid. Action: Set SJREJDVT to hexadecimal zeros or to the value returned on the previous call.

Table 7. Return and Reason Codes for the SJFREQ RETRIEVE Service (continued)

Return Code	Reason Code	Meaning and Action
4	005	<p>Meaning: System error. The information referenced by parameter field SJREJDVT does not exist.</p> <p>Action: SJF did not initialize properly. Contact the appropriate IBM support personnel.</p>
4	0C8	<p>Meaning: Program error. SJRETOKN refers to a JCL statement that is not valid.</p> <p>Action: Supply a value in SJRETOKN that refers to a valid JCL statement.</p>
4	0C9	<p>Meaning: Program error. A keyword in the keyword list is not defined to the JCL statement referred to by SJRETOKN.</p> <p>Action: SJREKERR contains the address of the keyword in error. Correct or delete the keyword in error.</p>
4	258	<p>Meaning: Program error. The area specified by SJRESIZE is less than the minimum allowed.</p> <p>Action: Define SJRESIZE to a size that can contain at least one text unit.</p>
4	25B	<p>Meaning: Program error. No storage area address was specified.</p> <p>Action: Specify a storage area address in SJREAREA.</p>
4	25C	<p>Meaning: Program error. The value in SJREKWD indicates that no keywords were specified.</p> <p>Action: Specify a value of one or greater for SJREKWD.</p>
4	25D	<p>Meaning: Program error. No keyword list address was specified.</p> <p>Action: Specify a keyword list address in SJREKWDL.</p>
4	25F	<p>Meaning: Program error. Zero is given as the value for a keyword in the keyword list.</p> <p>Action: SJREKERR contains the address of the keyword. Change the zero to a valid value.</p>
0C	000	<p>Meaning: System error. The system could not obtain storage for this request.</p> <p>Action: Inform your system programmer of this problem.</p>
10	000	<p>Meaning: System error. The ESTAE-type recovery routine failed.</p> <p>Action: Inform your system programmer of this problem.</p>
14	000	<p>Meaning: System error. SJF encountered a condition that caused an abnormal termination.</p> <p>Action: Check the input parameters, particularly any pointer fields, to determine if the input values are correct.</p>

SJFREQ Macro

Table 7. Return and Reason Codes for the SJFREQ RETRIEVE Service (continued)

Return Code	Reason Code	Meaning and Action
18	000	Meaning: System error. The service routines for SJFREQ are not available. Action: SJF did not initialize properly. Contact the appropriate IBM support personnel.

SJFREQ SWBTU_MERGE Service

Use the SWBTU_MERGE service to merge a list of base SWBTUs with a list of merge SWBTUs that contain modifications. An application can also use the SWBTU_MERGE service to indicate keys to be removed from a list of SWBTUs.

Programming Requirements

You must provide input information in the SWBTU_MERGE parameter list (IEFSJSMP).

In addition to the IEFSJSMP parameter list, you can provide an erase list. An erase list is a contiguous set of text units that you request to be erased from a base SWBTU. An erase list is required when you set SJSMETUP and SJSMETUS. If only a merge SWBTU and erase list are provided on input, SWBTU_MERGE validates the erase list, but does not apply it to the resulting SWBTU.

SJFREQ SWBTU_MERGE Input Parameters

For each SJFREQ invocation, you need to initialize certain fields of parameter list IEFSJSMP. The list of parameters and descriptions of their values are below.

SJSMID	The identifier 'SJS' of the SWBTU_MERGE parameter list. Assign the symbolic equate SJSMLGTH to this field.
SJSMVERS	The current version number of the SWBTU_MERGE parameter list. Assign the symbolic equate SJSMLGTH to this field.
SJSMFLAG	The environment control flag.
SJSMNOCU	Indicates whether the SJF environment is preserved from call to call or obtained for each call. Set to one to preserve the environment from call to call. Set to zero to obtain a new environment for a call.
SJSMLLEN	The length of the SWBTU_MERGE parameter list (IEFSJSMP). Assign the symbolic equate SJSMLGTH to this field.
SJSMSTOR	The local working storage pointer or zero. This field must contain zero on the first SWBTU_MERGE call. On subsequent calls, the field contains the value returned from the most recent SWBTU_MERGE call. Use the value returned for subsequent calls.
SJSMAREA	The address of the area to which a single SWBTU is returned after SWBTU_MERGE processing. SJSMSIZE contains the size of this area.
SJSMSIZE	The length of the area to which a single SWBTU is returned after SWBTU_MERGE processing. Specify the size of SJSMAREA in this field.
SJSMWBN	The number of SWBTU pointers in the base SWBTU pointer list.

SJFREQ Macro

This field can be zero if the address of the base SWBTU pointer list (SJSMSWBA) is also zero and you specify a merge SWBTU pointer list.

SJSMSWBA	The address of the base SWBTU pointer list. You can specify zero for this value if the number of SWBTU pointers in the base SWBTU pointer list (SJSMSWBN) is zero and you specify a merge SWBTU pointer list.
SJSMMTUP	The address of the merge SWBTU pointer list. You can specify zero for this value if the number of SWBTU pointers in the merge SWBTU pointer list (SJSMMTUN) is zero and you specify a base SWBTU pointer list.
SJSMMTUN	The number of SWBTU pointers in the merge SWBTU pointer list. You can specify zero for this value if the address of the merge SWBTU pointer list (SJSMMTUP) is zero and you specify a base SWBTU pointer list.
SJSMETUS	The number of elements contained on input in the erase text unit list area. You can specify zero for this value if the address of the erase text unit list area (SJSMETUP) is zero.
SJSMETUP	The address of the erase text unit list area. You can specify zero for this value if the address of the erase text unit list area (SJSMETUS) is zero.
SJSMJDVT	Enter zero for this field on input. This field must contain zero on the first SWBTU_MERGE call. On subsequent calls, the field contains the value returned from the most recent SWBTU_MERGE call. Use the value returned for subsequent calls.
SJSMWARN	Set on to indicate that processing should continue after allowable errors. Clear this field so that processing does not continue after allowable errors. Allowable errors are described the section “Merging SWBTUs” in <i>z/OS MVS Programming: Authorized Assembler Services Guide</i> .
SJSMBYMV	Set on to indicate whether the merge SWBTU text units should be validated.
SJSMBYEV	Set on to indicate whether the erase text units’ keys should be validated.
SJSMSBTL	When SJSMSWBA and SJSMSWBN contain values, this field must contain the mapping of the base SWBTU pointer list. When SJSMMTUP and SJSMMTUN contain values, this field must contain the mapping of the merge SWBTU pointer list.

SJFREQ SWBTU_MERGE Output Parameters

Data for an SJFREQ SWBTU_MERGE function is returned in several fields of the IEFSJSMP parameter list. The list of output parameters and their descriptions are below.

SJSMREAS	Contains a reason code returned from the SWBTU_MERGE service. This field contains a value when register 15 contains a return code other than zero.
SJSMAREA	Contains a pointer to the output SWBTU.
SJSMTULN	Contains the size of the output SWBTU. The size is the total size of the prefix and all the text units.

SJFREQ Macro

SJSMMKER	If a validation error occurred, this field contains the key from the first merge SWBTU text unit in the merge SWBTU list where SWBTU_MERGE encountered an error. Otherwise, this field contains zero.
SJSMEKER	If a validation error occurred, this field contains the key from the first erase SWBTU in the erase SWBTU list where SWBTU_MERGE encountered the error. Otherwise, this field contains zero.
SJSMJDVT	If a value is returned in this field, use the returned value in this field for subsequent uses of the service.
SJSMRETC	The return code from the service in which the error was encountered.
SJSMERRS	This field contains a value when certain errors occur. See Table 9 on page 69 for descriptions of the errors.
SJSMERRP	The point in the base or merge SWBTU pointer list entry at which an error was encountered.

SJFREQ SWBTU_MERGE ABEND Codes

SJFREQ SWBTU_MERGE might abnormally terminate with abend code X'054'. See *z/OS MVS System Codes* for an explanation and programmer response for this code.

SJFREQ SWBTU_MERGE Return and Reason Codes

SJFREQ SWBTU_MERGE return codes appear in register 15. When SJFREQ returns control to your program, SJSMREAS contains a reason code if register 15 contains a nonzero value. Return and reason codes are defined in macro IEFSJRC. The following table identifies the hexadecimal return and reason code combinations, tells what each means, and recommends an action you should take. Table 9 on page 69 lists and describes additional reason codes that appear in output field SJSMERRS when certain errors occur.

Table 8. Return and Reason Codes for the SJSMREAS Macro

Return Code	Reason Code	Meaning and Action
0	000	Meaning: SWBTU_MERGE processing completed successfully. Action: None.
0	0CA	Meaning: Program error. SWBTU_MERGE processing completed successfully; however, the value in SJSMMKER or SJSMEKER was ignored. SJSMMKER or SJSMEKER contains a key in error. Action: Determine if it is acceptable for the information in SJSMMKER or SJSMEKER to be ignored.
0	0CB	Meaning: Program error. SWBTU_MERGE processing completed successfully; however, the text unit count for the key that appears in SJSMMKER is not valid. All data for this text unit is ignored. Action: Determine if it is acceptable for the text unit data to be ignored.

Table 8. Return and Reason Codes for the SJSMMREAS Macro (continued)

Return Code	Reason Code	Meaning and Action
0	1F4	<p>Meaning: Program error. SWBTU_MERGE processing completed successfully; however, a text unit length in a length-value pair for the key that appears in SJSMMKER is not valid. All data for this text unit is ignored.</p> <p>Action: Check the length specified for the key that appears in SJSMMKER. Determine if it is acceptable for the text unit to be ignored.</p>
0	1F5	<p>Meaning: Program error. SWBTU_MERGE processing completed successfully; however, a text unit value in a length-value pair for the key that appears in SJSMMKER is not valid. All data for this text unit is ignored.</p> <p>Action: Check the value specified for the key that appears in SJSMMKER. Determine if it is acceptable for the text unit to be ignored.</p>
0	1F6	<p>Meaning: Program error. SWBTU_MERGE processing completed successfully; however, a numeric value of a length-value pair in a text unit exceeds the maximum allowable value. SJSMMKER contains the key from the text unit that is in error. All data for this text unit is ignored.</p> <p>Action: Check the value specified in the text unit and determine if it is acceptable for the text unit to be ignored.</p>
0	1F7	<p>Meaning: Program error. SWBTU_MERGE processing completed successfully; however, a numeric value of a length-value pair in a text unit is less than the minimum allowable value. SJSMMKER contains the key from the text unit that is in error. All data for this text unit is ignored.</p> <p>Action: Check the value specified in the text unit and determine if it is acceptable for the text unit to be ignored.</p>
0	1FE	<p>Meaning: Program error. SWBTU_MERGE processing completed successfully; however, a character value level length in a text unit exceeds the maximum allowed for character data. SJSMMKER contains the key from the text unit that is in error. All data for this text unit is ignored.</p> <p>Action: Check the value specified in the text unit and determine if it is acceptable for the text unit to be ignored.</p>
0	1FF	<p>Meaning: Program error. SWBTU_MERGE processing completed successfully; however, the number of levels for a character value in a text unit exceeds the maximum allowed. SJSMMKER contains the key from the text unit that is in error. All data for this text unit is ignored.</p> <p>Action: Check the value specified in the text unit and determine if it is acceptable for the text unit to be ignored.</p>

Table 8. Return and Reason Codes for the SJSMMREAS Macro (continued)

Return Code	Reason Code	Meaning and Action
0	200	<p>Meaning: Program error. SWBTU_MERGE processing completed successfully; however, the first character of the level in the character value in a text unit is not valid for character data. SJSMMKER contains the key from the text unit that is in error. All data for this text unit is ignored.</p> <p>Action: Check the value specified in the text unit and determine if it is acceptable for the text unit to be ignored.</p>
0	201	<p>Meaning: Program error. SWBTU_MERGE processing completed successfully; however, a character is not valid for character data. The incorrect value is not the first character of the level in the character value in a text unit. SJSMMKER contains the key from the text unit that is in error. All data for this text unit is ignored.</p> <p>Action: Check the value specified in the text unit and determine if it is acceptable for the text unit to be ignored.</p>
0	226	<p>Meaning: Program error. SWBTU_MERGE processing completed successfully; however, a text unit contained a text character that is outside the range X'40' - X'FE'. SJSMMKER contains the key from the text unit that is in error. All data for this text unit is ignored.</p> <p>Action: Check the value specified in the text unit and determine if it is acceptable for the text unit to be ignored.</p>
0	227	<p>Meaning: SWBTU_MERGE processing completed successfully; however, the sequence of characters in a text unit is not valid. SJSMMKER contains the key from the text unit that is in error. All data for this text unit is ignored.</p> <p>Action: Check the value specified in the text unit parameter and determine if it is acceptable for the text unit to be ignored.</p>
0	228	<p>Meaning: SWBTU_MERGE processing completed successfully; however, there are bits on in the text unit parameter that the system does not recognize. SJSMMKER contains the key from the text unit that is in error. All data for this text unit is ignored.</p> <p>Action: Check the value specified in the text unit parameter and determine if it is acceptable for the text unit to be ignored.</p>
0	76C	<p>Meaning: SWBTU_MERGE processing completed successfully. The resulting SWBTU (pointed to by SJSMMAREA) has a prefix and no text units.</p> <p>Action: None required. However, you might take some action based on your application.</p>
4	0C8	<p>Meaning: Program error. The specified verb is not defined to the system.</p> <p>Action: Specify a valid verb.</p>

Table 8. Return and Reason Codes for the SJSMMREAS Macro (continued)

Return Code	Reason Code	Meaning and Action
4	0CA	<p>Meaning: Program error. The system does not recognize the text unit key. SJSMMKER or SJSMEKER contains the key from the text unit that is in error. All data for this text unit is ignored.</p> <p>Action: Check the text unit key. Correct the key and issue the SJFREQ macro again.</p>
4	0CB	<p>Meaning: Program error. The count in a text unit is not valid. SJSMMKER contains the key for the text unit that is not valid.</p> <p>Action: Check the text unit count. Correct the count and issue the SJFREQ macro again.</p>
4	1F4	<p>Meaning: Program error. A length in a length-value pair in a text unit is not valid. SJSMMKER contains the key for the text unit that is not valid.</p> <p>Action: Check the length. Correct the length and issue the SJFREQ macro again.</p>
4	1F5	<p>Meaning: Program error. A text unit value in a length-value pair is not valid. SJSMMKER contains the key for the text unit that is not valid.</p> <p>Action: Check the value. Correct the value and issue the SJFREQ macro again.</p>
4	1F6	<p>Meaning: Program error. A numeric value of a length-value pair in a text unit exceeds the maximum allowable value. SJSMMKER contains the key from the text unit that is in error.</p> <p>Action: Check the value. Correct the value and issue the SJFREQ macro again.</p>
4	1F7	<p>Meaning: Program error. A numeric value of a length-value pair in a text unit is less than the minimum allowable value. SJSMMKER contains the key from the text unit that is in error.</p> <p>Action: Check the value. Correct the value and issue the SJFREQ macro again.</p>
4	1FE	<p>Meaning: Program error. A character value level length in a text unit exceeds the maximum allowed for character data. SJSMMKER contains the key from the text unit that is in error.</p> <p>Action: Check the value specified in the text unit. Correct the value and issue the SJFREQ macro again.</p>
4	1FF	<p>Meaning: Program error. The number of levels for a character value in a text unit exceeds the maximum allowed. SJSMMKER contains the key from the text unit that is in error.</p> <p>Action: Check the value specified in the text unit. Correct the value and issue the SJFREQ macro again.</p>

SJFREQ Macro

Table 8. Return and Reason Codes for the SJSMREAS Macro (continued)

Return Code	Reason Code	Meaning and Action
4	200	<p>Meaning: Program error. A character in a text unit is not valid for character data. The character is the first character in the level in the text unit. SJSMMKER contains the key from the text unit that is in error.</p> <p>Action: Check the value specified in the text unit. Correct the value and issue the SJFREQ macro again.</p>
4	201	<p>Meaning: Program error. A character is not valid for character data. The character is not the first character of the level in the character value in a text unit. SJSMMKER contains the key from the text unit that is in error.</p> <p>Action: Check the value specified in the text unit. Correct the value and issue the SJFREQ macro again.</p>
4	206	<p>Meaning: A text unit key contained hex zeros.</p> <p>Action: Check the text unit key value specified in the text unit. Correct the value and issue the SJFREQ macro again.</p>
4	226	<p>Meaning: Program error. A text unit contained a text character that is outside the range X'40' - X'FE'. SJSMMKER contains the key from the text unit that is in error.</p> <p>Action: Check the value specified in the text unit. Correct the value and issue the SJFREQ macro again.</p>
4	227	<p>Meaning: The sequence of characters in a text unit is not valid. SJSMMKER contains the key from the text unit that is in error.</p> <p>Action: Check the value specified in the text unit parameter. Correct the value and issue the SJFREQ macro again.</p>
4	228	<p>Meaning: There are bits on in the parameter that the system does not recognize. SJSMMKER contains the key from the text unit that is in error.</p> <p>Action: Check the value specified in the text unit parameter. Correct the value and issue the SJFREQ macro again.</p>
4	76D	<p>Meaning: Program error. The output area (identified by SJSMAREA and SJSMSIZE) is not large enough for the resulting SWBTU.</p> <p>Action: Increase the size of the output area and issue the SJFREQ macro again.</p>

Table 8. Return and Reason Codes for the SJSMREAS Macro (continued)

Return Code	Reason Code	Meaning and Action
4	76E	<p>Meaning: Program error. One of the following occurred:</p> <ul style="list-style-type: none"> The base SWBTU pointer list address (SJSMWBBA) is zero and the number of SWBTU pointers (SJSMWBNA) in the base pointer list is not zero The SWBTU pointer list address is not zero and the number of SWBTU pointers in the base pointer list is zero. <p>Action: Supply the correct values for SJSMWBBA and SJSMWBNA and issue the SJFREQ macro again.</p>
4	76F	<p>Meaning: Program error. One of the following occurred:</p> <ul style="list-style-type: none"> The merge SWBTU pointer list address (SJSMMTUP) is zero and the number of SWBTU pointers (SJSMMTUN) in the merge pointer list is not zero The merge SWBTU pointer list address is not zero and the number of SWBTU pointers in the merge pointer list is zero. <p>Action: Supply the correct values for SJSMMTUP and SJSMMTUN and issue the SJFREQ macro again.</p>
4	770	<p>Meaning: Program error. One of the following occurred:</p> <ul style="list-style-type: none"> The erase text unit address (SJSMETUP) is zero and the size of the erase text unit list area (SJSMETUS) is not zero The erase text unit address (SJSMETUP) is not zero and the size of the erase text unit list area (SJSMETUS) is zero. <p>Action: Supply the correct values for SJSMETUP and SJSMETUS and issue the SJFREQ macro again.</p>
4	771	<p>Meaning: Program error. Either the output area address (SJSMAREA) is zero or the output area size (SJSMASIZE) is not greater than zero.</p> <p>Action: Check the values in SJSMAREA and SJSMASIZE and determine which is incorrect. Specify valid values and issue the SJFREQ macro again.</p>
4	772	<p>Meaning: Program error. One of the following happened:</p> <ul style="list-style-type: none"> Neither a base SWBTU nor a modify SWBTU were specified (SJSMWBBA and SJSMMTUP are both zero). A base SWBTU was provided, but no modify SWBTU and no erase list were provided (SJSMMTUP and SJSMETUP are zero). <p>Action: Do one of the following:</p> <ul style="list-style-type: none"> Specify a base or modify SWBTU. If a base SWBTU was provided, specify either a modify SWBTU or an erase list.

SJFREQ Macro

Table 8. Return and Reason Codes for the SJSMBREAS Macro (continued)

Return Code	Reason Code	Meaning and Action
4	773	<p>Meaning: Program error. The verb name in the merge SWBTU list does not match the verb name in the base SWBTU list.</p> <p>Action: Supply input for the same verb in both the base SWBTU list and the merge SWBTU list.</p>
4	774	<p>Meaning: Program error. One of the following occurred:</p> <ul style="list-style-type: none"> The version number (SJSMBERS) supplied in the parameter list does not match the version defined in macro IEFJSMBP The length (SJSMBLEN) supplied in the parameter list does not match the actual length. <p>Action: Ensure that the constant SJSMBVER is used in field SJSMBERS of the input parameter list and that the constant SJSMBLGTH is used in field SJSMBLEN of the input parameter list.</p>
4	7A1	<p>Meaning: Program error. The base SWBTU list contains an error.</p> <p>Action: Check output fields SJSMBERP and SJSMBERS. SJSMBERP contains the address of a SWBTU that is in error. SJSMBERS may contain a reason code that indicates an error in the base or merge SWBTU list. See Table 9 on page 69 for the reason code descriptions.</p>
0C	000	<p>Meaning: System error. The system could not obtain storage for this request.</p> <p>Action: Inform your system programmer of this problem.</p>
10	000	<p>Meaning: System error. The ESTAE-type recovery routine failed.</p> <p>Action: Inform your system programmer of this problem.</p>
14	000	<p>Meaning: System error. SJF encountered a condition that caused an abnormal termination.</p> <p>Action: Check the input parameters, particularly any pointer fields, to determine if the input values are correct.</p>
18	000	<p>Meaning: System error. The service routines for SJFREQ are not available.</p> <p>Action: SJF did not initialize properly. Contact the appropriate IBM support personnel.</p>

Output field SJSMBERS contains a reason code when certain errors occur. Table 9 on page 69 lists the reason codes and their meanings.

Table 9. Return and Reason Codes for the SJFREQ Macro SWBTU_MERGE Service

Return Code	Reason Code	Meaning and Action
4	018	<p>Meaning: Program error. The application did not supply either the base SWBTU list or the merge SWBTU list.</p> <p>Action: Determine which list is missing. Supply the missing list and issue the SJFREQ macro again.</p>
4	019	<p>Meaning: Program error. The prefix for either the base SWBTU list or merge SWBTU list is not valid.</p> <p>Action: Determine which prefix is not valid. Supply the correct prefix and issue the SJFREQ macro again.</p>
4	028	<p>Meaning: Program error. The verb and label values in either the base SWBTU list or the merge SWBTU list do not match.</p> <p>Action: Determine which pair does not match. Correct the information and issue the SJFREQ macro again.</p>

SJFREQ VERIFY Service

Use VERIFY to validate and build text units to represent an OUTDES statement, its operands, subparameters and sublist elements. The OUTDES information corresponds to SJF-defined information on the OUTPUT JCL statement. If you do not specify correctly the input subparameter data, VERIFY returns an error message.

SJFREQ VERIFY Input Parameters

For each SJFREQ invocation, you need to initialize certain fields of parameter list IEFSJVEP. See SJVEP in *z/OS MVS Data Areas, Vol 4 (RD-SRRA)* for the mapping provided by the IEFSJVEP mapping macro. The list of parameters and descriptions of their values are below, followed by Table 10 that summarizes the required parameter fields for the three SJFREQ VERIFY functions.

VERIFY performs three functions. Use Table 10 to determine the required parameters for the SJFREQ VERIFY function you want to request. The table gives details about the required values.

- SJVEID** The identifier 'SJVE' of the VERIFY parameter list. Assign the symbolic equate SJVECID to this field.
- SJVEVERS** The current version number of the SJFREQ VERIFY parameter list. Assign the symbolic equate SJVECVVER to this field.
- SJVEFLAG** The environment control flag.
 - SJVENOCU** Indicates whether the SJF environment is preserved from call to call or obtained for each call. Parameter SJVESTOR references the environment. When set to 0, the environment is obtained for each call. When set to 1, the environment is preserved.
 - SJVEUNAU** Indicates whether the caller is unauthorized. If an

SJFREQ Macro

application makes repetitive service calls using the same SJF environment, the authorization must be same for each call.

SJVELEN	The length of the VERIFY parameter list (IEFSJVEP). Assign the symbolic equate SJVELGTH to this field.
SJVESTOR	The local working storage pointer or zero. This field must contain zero on the first VERIFY call. On subsequent calls, the field contains the value returned from the most recent VERIFY call.
SJVEJDVT	Enter zero for this field on input. If a value is returned in the field, use the returned value for subsequent uses of the service.
SJVECMND	The name of the statement that contains the output descriptor information. Specify the statement name OUTDES in an 8-character field, left-justified, and padded with blanks.
SJVEOPEP	The pointer to the operand or keyword operand to be validated. This field should contain an address of the first byte of the operand or keyword operand. Set the length of the operand in SJVEOPEL. Zero is a valid value when specifying just the statement name as specified in SJVECMND.
SJVEOPEL	The length of the operand or keyword operand to be validated. This field should contain the actual length of the operand or keyword operand pointed to by SJVEOPEP.
SJVEPARM	The subparameter number representing the subparameter to be validated. Set this value to one whenever validating an operand. When validating a keyword operand, set this to the correct subparameter number.
SJVESUBL	The sublist element number representing the sublist element to be validated. This field should be set to zero if the subparameter is not a sublist.
SJVEPRMP	The pointer to the subparameter or sublist element to be validated. Set the length of the subparameter or sublist element in SJVEPRML.
SJVEPRML	The length of the subparameter or sublist element pointed to by SJVEPRMP. This field should contain the actual length of the subparameter or sublist element pointed to by SJVEPRMP.
SJVETUBL	The length of the text unit output area. The field should contain the length of the output area pointed to by SJVETUBP. The minimum you should specify for the output area is 256 bytes plus sufficient storage to accommodate the text units that will be built as a result of your VERIFY request. A work area of 1K is large enough for any set of text units to be built.
SJVETUBP	The pointer to the text unit output area. The output area will contain the text unit pointer list and text units upon return from the VERIFY service call. If the output area is not large enough to hold all the text units and the text unit pointer list, you can point to and use an additional output area. When using additional output areas, the original output area must be accessible to SJF; it cannot be freed or changed. Set the length of the output area in SJVETUBL.
SJVEFLG1	The VERIFY option flag.
SJVELSTC	Indicates last call for the text units being built. Set

this indicator to one for the last VERIFY call. Setting this indicator on makes the text units available for use.

SJVERSBS

Indicates that the same text unit output area is to be used for multiple calls. SJVETUBS contains the returned length of the area the service used to build the text units and text unit pointer list. When this indicator is on, SJVETUBP must be the same for each VERIFY call.

SJVEQUOT

Indicates that a subparameter was specified in quotes. To allow quotes on all subparameters, the caller can set this bit to zero. Note that some subparameters do not allow quotes. For any value specified in quotes, the caller should have:

- Removed the delimiting quotes
- Converted two consecutive single quotes to one single quote.

SJVEPRFX

The prefix to be concatenated to a subparameter that is a data set name. This field is not used if SJVEQUOT is on or if the subparameter is not defined as allowing unqualified data set names. If no prefix is specified, this field must be set to zero.

Table 10. Required Fields for SJFREQ VERIFY Functions

IEFSJVEP parameter list field	Validate a subparameter or sublist element and build a text unit	Validate an operand or keyword operand	Validate a statement name
SJVEID	Symbolic equate SJVECID	Symbolic equate SJVECID	Symbolic equate SJVECID
SJVEVERS	Symbolic equate SJVECV	Symbolic equate SJVECV	Symbolic equate SJVECV
SJVENOCU	Set to indicate if the SJF environment should be reused.	Set to indicate if the SJF environment should be reused.	Set to indicate if the SJF environment should be reused.
SJVEUNAU	Set to indicate if the caller is authorized.	Set to indicate if the caller is authorized.	Set to indicate if the caller is authorized.
SJVELEN	Symbolic equate SJVELGTH	Symbolic equate SJVELGTH	Symbolic equate SJVELGTH
SJVESTOR	Local working storage pointer or zero	Local working storage pointer or zero	Local working storage pointer or zero
SJVECMND	Statement name associated with the operand for which the subparameter or sublist is to be validated.	Statement name associated with the operand to be validated.	Statement name to be validated.
SJVEOPEP	Address of the field that contains the subparameter or sublist element to be validated.	Address of the field that contains the operand or keyword operand to be validated.	zeros
SJVEOPEL	Length of the field that contains the subparameter or sublist element (SJVEOPEP).	Length of the field that contains the operand or keyword operand to be validated (SJVEOPEP).	not used

SJFREQ Macro

Table 10. Required Fields for SJFREQ VERIFY Functions (continued)

IEFSJVEP parameter list field	Validate a subparameter or sublist element and build a text unit	Validate an operand or keyword operand	Validate a statement name
SJVEPARM	Number of the subparameter to be verified (1 for the first, 2 for the second, . . .)	zeros	zeros
SJVESUBL	If the subparameter to be verified is a sublist, specify the number of the sublist element (1 for the first, 2 for the second, . . .). If the subparameter is not a sublist, specify zero.	not used	not used
SJVEPRMP	Address of the field with the subparameter or sublist to be verified.	not used	not used
SJVEPRML	Length of the subparameter or sublist to be verified.	not used	not used
SJVETUBL	Length of the SJFREQ VERIFY workarea.	not used	not used
SJVETUBP	Pointer to the SJFREQ VERIFY workarea.	not used	not used
SJVELSTC	Last call bit	not used	not used
SJVEQUOT	Subparameters can be specified in quotes.	not used	not used
SJVERSBS	Text unit buffer is to be used for multiple calls.	not used	not used
SJVEPRFX	Prefix to be concatenated to a subparameter.	not used	not used

SJFREQ VERIFY Output Parameters

Data for an SJFREQ VERIFY function is returned in several fields of the IEFSJVEP parameter list. The list of output parameters and their descriptions are below, followed by Table 11 on page 73 that summarizes the output parameters returned for each of the three SJFREQ VERIFY functions.

- SJVEREAS** The reason code returned from VERIFY processing. See Return and Reason Codes with Related Message Text for information about specific reason codes.
- SJVEJDVT** If the caller specified zero on input, this field contains the default JDVT name. Otherwise, the value in this field will be the same as it was on input.
- SJVETUBS** The length of the storage used in the text unit output area to build the text unit pointer list and text units. This field is filled in only when SJVERSBS is set on. The text unit output area is referenced by fields SJVETUBL and SJVETUBP.
- SJVETUPL** The pointer to the beginning of the text unit pointer list in the text unit output area.
- SJVEOPD** The SJF description of the operand or keyword operand that was referenced on input by the fields SJVEOPEP and SJVEOPEL. Applications can use information in this field in messages to their application users in place of the operand or keyword operand.

Refer to “Operand Descriptions” for the operands and their descriptions. VERIFY returns a value in this field for return code 0 and reason codes with return code 4.

- SJVEOPDL** The length of the operand or keyword operand description returned in field SJVEOPD.
- SJVEMSGL** The length of the message information returned in SJVEMSG.
- SJVEMSG** This field contains a message that indicates the correct syntax for the subparameter or sublist element that is in error. The subparameter or sublist element is referenced by the input fields SJVEPRMP and SJVEPRML. VERIFY returns a value in this field for return code 4 with some reason codes. Refer to “Return and Reason Codes with Related Message Text” on page 75 for the actual message text VERIFY returns.

When an application receives a return code of 0 or 4, VERIFY returns values in the parameter list fields indicated below. The table is organized by function and return code. VERIFY does not fill in any output parameter list fields when it returns a return code 8 or above.

Table 11. SJFREQ VERIFY Output Fields

IEFSJVEP parameter list field	Validate a subparameter or sublist element and build a text unit Return Code 0	Validate a subparameter or sublist element and build a text unit Return Code 4	Validate an operand or keyword operand Return Code 0	Validate an operand or keyword operand Return Code 4	Validate a statement name Return Code 0	Validate a statement name Return Code 4
SJVEREAS	Value returned	Value returned	Value returned	Value returned	Value returned	Value returned
SJVEJDVT	Value returned	Value returned	Value returned	Value returned	Value returned	Value returned
SJVEUBS	Value returned only when SJVERSBS is set to 1.	—	—	—	—	—
SJVEUPL	Value returned	Value returned for some reason codes.	—	—	—	—
SJVEOPD	Value returned	Value returned for some reason codes.	Value returned	—	—	—
SJVEOPDL	Value returned	Value returned for some reason codes.	Value returned	—	—	—
SJVEMSGL	—	Value returned for some reason codes.	—	—	—	—
SJVEMSG	—	Value returned for some reason codes.	—	—	—	—

Operand Descriptions

Operand descriptions appear in parameter list field SJVEOPD as indicated in Table 11. The table that follows lists the operands, keyword operands and their

SJFREQ Macro

descriptions.

Table 12. SJF Operand and Keyword Operand Descriptions. These descriptions appear in parameter field SJVEOPD.

Operand or Keyword Operand	Description
ADDRESS	ADDRESS FOR SEPARATOR PAGE
BUILDING	BUILDING ID
BURST	BURSTER TRIMMER STACKER
NOBURST	BURSTER TRIMMER STACKER
CHARS	CHARACTER ARRANGEMENT TABLE
CKPTLINE	CHECKPOINT LINES
CKPTPAGE	CHECKPOINT PAGES
CKPTSEC	CHECKPOINT SECONDS
CLASS	OUTPUT CLASS
COMPACT	COMPACTION TABLE NAME
CONTROL	CARRIAGE CONTROL
COPIES	NUMBER OF COPIES
DATAACK	DATA CHECK
DEFAULT	DEFAULT OUTPUT DESCRIPTOR
NODEFAULT	DEFAULT OUTPUT DESCRIPTOR
DEPT	DEPARTMENT ID
DEST	OUTPUT DESTINATION
DPAGELBL	DATA PAGE LABEL
NODPAGELBL	DATA PAGE LABEL
FCB	FORMS CONTROL IMAGE
FLASH	FORMS OVERLAY
FORMDEF	FORM DEFINITION MEMBER NAME
FORMS	PRINT FORMS
GROUPID	OUTPUT GROUP IDENTIFIER
INDEX	RIGHT PRINT POSITION INDEX
LINDEX	LEFT PRINT POSITION INDEX
LINECT	LINE COUNT
MODIFY	COPY MODIFICATION MODULE
NAME	NAME OF SYSOUT OWNER
NOTIFY	DESTINATION FOR PRINT COMPLETE MESSAGES
OUTBIN	PRINTER OUTPUT BIN ID
OUTDISP	SYSOUT DISPOSITION
PAGEDEF	PAGE DEFINITION MEMBER NAME
PIMSG	PRINTER INFORMATION MESSAGES
PRMODE	PROCESS MODE
PRTY	OUTPUT PRIORITY
ROOM	ROOM IDENTIFICATION
SYSAREA	SYSTEM PRINTABLE AREA
NOSYSAREA	SYSTEM PRINTABLE AREA
THRESHLD	MAXIMUM LINES OF OUTPUT
TITLE	NAME FOR SEPARATOR PAGE
TRC	TABLE REFERENCE CHARACTER

Table 12. SJF Operand and Keyword Operand Descriptions (continued). These descriptions appear in parameter field SJVEOPD.

Operand or Keyword Operand	Description
NOTRC	TABLE REFERENCE CHARACTER
UCS	UNIVERSAL CHARACTER SET
USERLIB	USER SPECIFIED AFP RESOURCE LIBRARIES
WRITER	EXTERNAL WRITER NAME

ABEND Codes

SJFREQ VERIFY might abnormally terminate with abend code X'054'. See *z/OS MVS System Codes* for an explanation and programmer response.

Return and Reason Codes with Related Message Text

SJFREQ VERIFY return codes appear in register 15. When SJFREQ returns control to your program, SJVEREAS contains a reason code. Return and reason codes are defined in macro IEFSJRC. The following table identifies the hexadecimal return and reason code combinations, tells what each means, and recommends an action you should take. Message text returned in field SJVEMSG appears with the meanings and actions for the related reason codes. *Italics* indicates variable parts of the message. Words in bold print indicate that VERIFY selects among those choices before returning the message.

For return code 4, reason codes 1F4 - 204, 226, and 4B3, the subparameter or sublist element referenced in SJVEPRMP and SJVEPRML is not correct. For each of these conditions, a description of the correct specification of the field is returned in SJVEMSG. The related message text appears with the appropriate reason codes that follow. SJVEOPD contains a description of the operand or keyword operand. Table 12 on page 74 lists the operands and keyword operands and their descriptions.

Table 13. Return and Reason Codes for the SJFREQ Macro VERIFY Service

Return Code	Reason Code	Meaning and Action
0	000	Meaning: VERIFY processing completed successfully. Action: None.
04	004	Meaning: Program error. The VERIFY request was not processed. The value specified in SJVEJDVT is not valid. Action: Set SJVEJDVT to hexadecimal zeros or to the value returned on the previous call.
04	005	Meaning: System error. The information referenced by parameter field SJVEJDVT does not exist. Action: SJF did not initialize properly. Contact the appropriate IBM support personnel.
04	0CB	Meaning: Program error. SJF does not recognize the subparameter specified in SJVEPARM. Action: Check the number of subparameters allowed and the value specified in SJVEPARM.

Table 13. Return and Reason Codes for the SJFREQ Macro VERIFY Service (continued)

Return Code	Reason Code	Meaning and Action
04	0CF	<p>Meaning: Program error. The command specified in SJVECMND is not recognized by SJF.</p> <p>Action: Check the spelling and specification (left-justified, padded with blanks) of the value specified in SJVECMND.</p>
04	0D0	<p>Meaning: Program error. The operand or keyword operand indicated by fields SJVEOPEP and SJVEOPEL is not recognized by SJF.</p> <p>Action: Check the specified length and the spelling of the operand or keyword operand. Make sure SJVEOPEP is a pointer value.</p>
04	1F4	<p>Meaning: Program error. The length of a specified subparameter or sublist element as specified in SJVEPRML is not valid.</p> <p>Action: Check the subparameter or sublist element length specified in SJVEPRML and the allowable subparameter or sublist element length.</p> <p>Message text:</p> <p>VALUE MUST BE 1 CHARACTER</p> <p>VALUE MUST BE <i>n</i> CHARACTERS</p> <p>VALUE MUST BE <i>minimum length</i> TO <i>maximum length</i> CHARACTERS</p> <p>If an error occurs involving a subparameter that is a data set name and the application specified a prefix, and SJVEQUOT is off (0), the following message might be added to the above message text.</p> <p>VALUE NOT ENCLOSED IN QUOTES IS CONCATENATED TO PREFIX <i>value</i> in <i>SJVEPRFX</i>.</p>
04	1F5	<p>Meaning: Program error. A subparameter that was specified is not a valid choice.</p> <p>Action: Specify one of the allowable subparameters for the keyword operand.</p> <p>Message text:</p> <p>VALUE MUST BE <i>choice1</i>. . .OR <i>choice n</i></p>
04	1F6	<p>Meaning: Program error. A numeric subparameter or sublist element exceeds the maximum allowable value.</p> <p>Action: Check the values specified in SJVEPRMP and SJVEPRML and the allowable value for this subparameter or sublist element.</p> <p>Message text:</p> <p>VALUE MUST BE IN THE RANGE OF <i>minimum value</i> TO <i>maximum value</i></p>

Table 13. Return and Reason Codes for the SJFREQ Macro VERIFY Service (continued)

Return Code	Reason Code	Meaning and Action
04	1F7	<p>Meaning: Program error. A numeric subparameter or sublist element is less than the minimum allowable value.</p> <p>Action: Check the values specified in SJVEPRMP and SJVEPRML and the allowable value for this subparameter or sublist element.</p> <p>Message text:</p> <p>VALUE MUST BE IN THE RANGE OF <i>minimum value</i> TO <i>maximum value</i></p>
04	1FA	<p>Meaning: Program error. No subparameter or sublist element was specified for the keyword operand.</p> <p>Action: Specify a value in SJVEPRML.</p> <p>Message text:</p> <p>VALUE MUST BE 1 LEVEL</p> <p>VALUE MUST BE 1 TO <i>maximum number of levels</i> LEVELS</p> <p>VALUE MUST BE 1 TO <i>maximum number of levels</i> LEVELS WITH 1 TO <i>maximum level length</i> CHARACTERS IN EACH LEVEL</p> <p>VALUE MUST BE 1 CHARACTER</p> <p>VALUE MUST BE <i>n</i> CHARACTERS</p> <p>VALUE MUST BE <i>minimum length</i> TO <i>maximum length</i> CHARACTERS</p> <p>If an error occurs involving a subparameter that is a data set name and the application specified a prefix, and SJVEQUOT is off (0), the following message might be added to the above message text.</p> <p>VALUE NOT ENCLOSED IN QUOTES IS CONCATENATED TO PREFIX <i>value in SJVEPRFX</i>.</p> <p>VALUE MUST BE <i>choice1</i>. . .OR <i>choice n</i></p> <p>VALUE MUST BE HEXADECIMAL CHARACTER, CHARACTERS</p> <p>VALUE MUST BE NUMERIC CHARACTER</p> <p>VALUE MUST BE NUMERIC CHARACTERS</p> <p>PRINTABLE CHARACTER, CHARACTERS</p> <p>VALUE MUST BE SPECIFIED WITH THE <i>keyword</i> KEYWORD</p>

Table 13. Return and Reason Codes for the SJFREQ Macro VERIFY Service (continued)

Return Code	Reason Code	Meaning and Action
04	1FE	<p>Meaning: Program error. The subparameter or sublist element level length exceeds the maximum allowed for character data.</p> <p>Action: Check the value in SJVEPRML.</p> <p>Message text:</p> <p>VALUE MUST BE 1 LEVEL</p> <p>VALUE MUST BE 1 TO <i>maximum number of levels</i> LEVELS</p> <p>VALUE MUST BE 1 TO <i>maximum number of levels</i> LEVELS WITH 1 TO <i>maximum level length</i> CHARACTERS IN EACH LEVEL</p> <p>If an error occurs involving a subparameter that is a data set name and the application specified a prefix, and SJVEQUOT is off (0), the following message might be added to the above message text.</p> <p>VALUE NOT ENCLOSED IN QUOTES IS CONCATENATED TO PREFIX <i>value in SJVEPRFX</i>.</p>
04	1FF	<p>Meaning: Program error. The number of levels for a subparameter or sublist element exceeds the maximum allowed.</p> <p>Action: Check the value pointed to by SJVEPRMP.</p> <p>Message text:</p> <p>VALUE MUST BE 1 LEVEL</p> <p>VALUE MUST BE 1 TO <i>maximum number of levels</i> LEVELS</p> <p>VALUE MUST BE 1 TO <i>maximum number of levels</i> LEVELS WITH 1 TO <i>maximum level length</i> CHARACTERS IN EACH LEVEL</p> <p>If an error occurs involving a subparameter that is a data set name and the application specified a prefix, and SJVEQUOT is off (0), the following message might be added to the above message text.</p> <p>VALUE NOT ENCLOSED IN QUOTES IS CONCATENATED TO PREFIX <i>value in SJVEPRFX</i>.</p>

Table 13. Return and Reason Codes for the SJFREQ Macro VERIFY Service (continued)

Return Code	Reason Code	Meaning and Action
04	200	<p>Meaning: Program error. The first character of the level in the subparameter or sublist element is not valid for character data.</p> <p>Action: Check the value pointed to by SJVEPRMP.</p> <p>Message text:</p> <p>VALUE MUST BE ALPHABETIC, NUMERIC, NATIONAL, OR <i>list of special characters</i> FIRST CHARACTER</p> <p>VALUE MUST BE ALPHABETIC, NUMERIC, NATIONAL, OR <i>list of special characters</i> CHARACTER</p> <p>VALUE MUST BE ALPHABETIC, NUMERIC, NATIONAL, OR <i>list of special characters</i> CHARACTERS</p> <p>VALUE CONTAINS INVALID FIRST CHARACTER</p> <p>If an error occurs involving a subparameter that is a data set name and the application specified a prefix, and SJVEQUOT is off (0), the following message might be added to the above message text.</p> <p>VALUE NOT ENCLOSED IN QUOTES IS CONCATENATED TO PREFIX <i>value in SJVEPRFX</i>.</p>
04	201	<p>Meaning: Program error. A character other than the first character of the level in the subparameter or sublist element is not valid for character data.</p> <p>Action: Check the value pointed to by SJVEPRMP.</p> <p>Message text:</p> <p>VALUE MUST BE ALPHABETIC, NUMERIC, NATIONAL OR, <i>list of special characters</i> CHARACTERS OTHER THAN THE FIRST</p> <p>VALUE MUST BE ALPHABETIC, NUMERIC, NATIONAL, OR <i>list of special characters</i> CHARACTER</p> <p>VALUE MUST BE ALPHABETIC, NUMERIC, NATIONAL, OR <i>list of special characters</i> CHARACTERS</p> <p>If an error occurs involving a subparameter that is a data set name and the application specified a prefix, and SJVEQUOT is off (0), the following message might be added to the above message text.</p> <p>VALUE NOT ENCLOSED IN QUOTES IS CONCATENATED TO PREFIX <i>value in SJVEPRFX</i>.</p>

Table 13. Return and Reason Codes for the SJFREQ Macro VERIFY Service (continued)

Return Code	Reason Code	Meaning and Action
04	202	<p>Meaning: Program error. The level specification is not valid.</p> <p>Action: Check the value pointed to by SJVEPRMP.</p> <p>Message text:</p> <p>VALUE MUST BE 1 LEVEL</p> <p>VALUE MUST BE 1 TO <i>maximum number of levels</i> LEVELS</p> <p>VALUE MUST BE 1 TO <i>maximum number of levels</i> LEVELS WITH 1 TO <i>maximum level length</i> CHARACTERS IN EACH LEVEL</p> <p>If an error occurs involving a subparameter that is a data set name and the application specified a prefix, and SJVEQUOT is off (0), the following message might be added to the above message text.</p> <p>VALUE NOT ENCLOSED IN QUOTES IS CONCATENATED TO PREFIX <i>value in SJVEPRFX</i>.</p>
04	203	<p>Meaning: Program error. Nonhexadecimal characters were specified for a subparameter or sublist element.</p> <p>Action: Check the value pointed to by SJVEPRMP.</p>
04	204	<p>Meaning: Program error. Nonnumeric characters were specified for a subparameter or sublist element.</p> <p>Action: Check the value pointed to by SJVEPRMP.</p> <p>Message text:</p> <p>VALUE MUST BE NUMERIC CHARACTER, CHARACTERS</p>
04	226	<p>Meaning: Program error. A text character that was specified for a subparameter or sublist element is not valid.</p> <p>Action: Check the value pointed to by SJVEPRMP.</p> <p>Message text:</p> <p>VALUE MUST BE PRINTABLE CHARACTER, CHARACTERS</p>
04	229	<p>Meaning: A keyword operand specified was not a valid choice.</p> <p>Action: Specify one of the allowable keyword operands.</p> <p>Message text:</p> <p>VALUE IS NOT AN ALLOWABLE KEYWORD OPERAND</p>

Table 13. Return and Reason Codes for the SJFREQ Macro VERIFY Service (continued)

Return Code	Reason Code	Meaning and Action
04	4B0	<p>Meaning: Program error. No statement was specified. The value specified for SJVECMND is zero.</p> <p>Action: Specify a statement in SJVECMND.</p>
04	4B1	<p>Meaning: Program error. No address was specified in SJVETUBP for the text unit output area.</p> <p>Action: Specify a value in SJVETUBP.</p>
04	4B2	<p>Meaning: Program error. There is not enough storage in the text unit output area to construct the text unit for the current value.</p> <p>Action: Increase the size of the output area, adjust SJVETUBP and SJVETUBL, and call VERIFY again with the same input.</p>
04	4B3	<p>Meaning: Program error. A subparameter or sublist element cannot be specified in quotes. SJVEQUOT is set on, but quotes are not allowed for the specified subparameter or sublist element.</p> <p>Action: Either set SJVEQUOT off or specify a value for which quotes are allowed.</p> <p>Message text:</p> <p>VALUE CANNOT BE SPECIFIED IN QUOTES</p>
04	4B4	<p>Meaning: Program error. The text unit output area is different than the text unit output area passed on the first call. SJVETUBP has a different value than on the first call, and SJVERSBS is set on. SJF cannot obtain the amount of contiguous storage necessary for the text unit pointer list and text units when more than one buffer is provided.</p> <p>Action: Check the value in SJVETUBP.</p>
08	000	<p>Meaning: Program error. The input parameter list, IEFSJVEP, is not valid.</p> <p>Action: Check to see if SJVEID, SJVEVER, or SJVELEN is incorrect and not recognized by SJF.</p>
0C	000	<p>Meaning: System error. The system could not obtain storage for this request.</p> <p>Action: Inform your system programmer of this problem.</p>
10	000	<p>Meaning: System error. The ESTAE-type recovery routine failed.</p> <p>Action: Inform your system programmer of this problem.</p>
14	000	<p>Meaning: System error. SJF encountered a condition that caused an abnormal termination.</p> <p>Action: Check the input parameters, particularly any pointer fields, to determine if the input values are correct.</p>

SJFREQ Macro

Table 13. Return and Reason Codes for the SJFREQ Macro VERIFY Service (continued)

Return Code	Reason Code	Meaning and Action
18	000	Meaning: System error. The service routines for SJFREQ are not available. Action: SJF did not initialize properly. Contact the appropriate IBM support personnel.

SJFREQ TERMINATE Service

TERMINATE frees the environment established by an SJF service. TERMINATE performs no other function.

SJFREQ TERMINATE Input Parameters

IEFSJREP, IEFSJSMP, or IEFSJVEP. The field SJVESTOR must contain a pointer to the SJF environment to be freed.

Return and Reason Codes

Return codes appear in register 15. The hexadecimal return codes from the SJFREQ TERMINATE service are as follows.

Table 14. Return Codes from the SJFREQ TERMINATE Service

Return Code	Meaning and Action
00	Meaning: TERMINATE processing completed successfully. Action: None
08	Meaning: Program error. The parameter list is not valid. Action: Check to see if SJVEID, SJVEVERS, or SJVELEN is incorrect and not recognized by SJF.
14	Meaning: Program or system error. An abnormal termination occurred. Action: Check the input parameters, particularly any pointer fields, to determine if the input values are correct.
18	Meaning: System error. The service routines for SJFREQ are not available. Action: SJF did not initialize properly. Contact the appropriate IBM support personnel.

Example

Invoke the VERIFY service to:

- Validate the syntax of the statement OUTDES, the keyword operand CHARS, and the subparameter (GT10)
- Build text units for the valid keyword operand and subparameter.

```
OUTDES out1 CHARS(GT10)
```

Use the label out1 for the OUTDES statement in this example. The statement is to be converted into text units and used as input to OUTADD, the dynamic output macro.

```
*****
*   This program segment has attributes that allow the defined   *
*   storage to be altered.                                       *
*****
*
```

```

*          Set up SJVEP, VERIFY parameter list area.          *
*                                                                *
*****
*
      XC    SJVEP(SJVELGTH),SJVEP  Clear the parameter list
      MVC   SJVEID,=A(SJVEPEYE)  Parameter list ID
      MVI   SJVEVERS,SJVECVVER    Parameter list version
      OI    SJVEFLAG,SJVENOCU     Indicate no cleanup to SJF on this
*                                call, another call to SJF is
*                                expected.
      LA    R4,SJVELGTH           Get parameter list length
      STH   R4,SJVELEN           Set parameter list length
*
*                                SJVESTOR and SJVEJDVT are properly
*                                set at zero from XC instruction.
*
*****
*          Set up statement and operand information.          *
*                                                                *
*****
*
      MVC   SJVECMND,STATMNT      Set statement name field to OUTDES
      ST    R2,SJVEOPEP           Set the operand pointer
      LR    R15,R3                Get pointer to last operand
*                                character
      SR    R15,R2                Get difference from first operand
*                                character
      LA    R15,1(R15)            Add 1 to get proper operand length
      STH   R15,SJVEOPEL         Set operand length
*
*****
*          Set up subparameter information.                    *
*                                                                *
*****
*
      LA    R15,1                Set up for first subparameter
      STC   R15,SJVEPARM         Set subparameter number to 1
      ST    R4,SJVEPRMP          Set the subparameter pointer
      LR    R15,R5                Get pointer to last subparameter
*                                character
      SR    R15,R4                Get difference from first
*                                subparameter character
      LA    R15,1(R15)            Add 1 to get proper subparameter
*                                length
      STH   R15,SJVEPRML         Set the subparameter pointer
*
*****
*          Set up output area information.                      *
*                                                                *
*****
*
      LA    R15,AREASIZE         Get output work area length
      STH   R15,SJVETUBL         Set text unit output area length
      LA    R15,OUTAREA          Get address of output work area
      ST    R15,SJVETUBP         Set text unit output area size
*
*****
*          Set up Register 1 to point to a word of storage that
*          contains the address of SJVEP.
*
*****
*
      LA    R4,SJVEP             Address of

```

SJFREQ Macro

```

        ST      R4,SJVEPPTR      the SJFREQ VERIFY
        LA      R1,SJVEPPTR      parameter list
*
*****
*
*          Invoke SJFREQ VERIFY service.
*
*****
*
*          SJFREQ REQUEST=VERIFY      Issue the SJF macro.
*
*****
*
*          Check for a zero return code.
*
*****
*
*          LTR    R15,R15          Check service return code
*          BNZ    SJFERR           Go to nonzero return processing
*
*
*
*****
*
*          In this portion of the example, a zero return and reason
*          code are received. The output fields from the service
*          contain the following information:
*
*          Register 15 - contains zero.
*
*          SJVEREAS - contains zero.
*
*          SJVEOPD - this field contains the operand description
*                   for CHARS:
*                   "CHARACTER ARRANGEMENT TABLE"
*
*          SJVEOPDL - this field contains the operand description
*                   length - decimal 27.
*
*          SJVETUPL - this field contains an address into OUTAREA
*                   that is the start of the text unit pointer
*                   list.
*
*          OUTAREA - this area contains the text unit pointer
*                   and the text unit that were built by VERIFY
*                   for the CHARS(GT10) specification.
*****
*
*
SJFERR DS      0H                Label used for branch when SJFREQ
*                                VERIFY returns with a nonzero
*                                return code.
*
*          Code to handle SJFREQ errors
*
*****
*
*          Storage definitions
*
*****
*
*          IEFSJVEP DSECT=NO      SJFREQ VERIFY parameter list area
*
*          SJVEPPTR DS      A      Field used to contain SJVEP address
*
*
*          OUTAREA DS      XL1024  Area used by SJFREQ VERIFY to build

```

```

*                               text units for valid operands and
*                               subparameters.
*
AREASIZE EQU  *-OUTAREA      Size of AREA
*
*****
*                               *
*      Equates and Constants      *
*                               *
*****
*
R0      EQU  0      Register 0
R1      EQU  1      Register 1
R2      EQU  2      Register 2
R3      EQU  3      Register 3
R4      EQU  4      Register 4
R5      EQU  5      Register 5
R6      EQU  6      Register 6
R7      EQU  7      Register 7
R8      EQU  8      Register 8
R9      EQU  9      Register 9
R15     EQU 15      Register 15
*
SJVEPEYE EQU  C'SJVE'      VERIFY parameter list identifier
STATMNT  DC   CL8'OUTDES '  Statement name
*
*****

```

SJFREQ Macro

SPIE — Specify Program Interruption Exit

Description

Note: IBM recommends that you use the ESPIE macro rather than SPIE. Callers in 31-bit addressing mode must use the ESPIE macro, which performs the same function as the SPIE macro for callers in both 24-bit and 31-bit addressing mode.

The SPIE macro specifies the address of an interruption exit routine and the program interruption types that are to cause the exit routine to get control.

Note: In MVS/370 the SPIE environment existed for the life of the task. In later versions of MVS, the SPIE environment is deleted when the request block that created it is deleted. That is, when a program running under a later version of MVS completes, any SPIE environments created by the program are deleted. This might create an incompatibility with MVS/SP Version 1 for programs that depend on the SPIE environment remaining in effect for the life of the task rather than the request block.

Each succeeding SPIE macro invocation completely overrides any previous SPIE macro specifications for the task. The specified exit routine is given control in the key of the TCB when one of the specified program interruptions occurs in any problem program of the task. When a SPIE exit routine issues the SPIE macro, the system resets (zeros) the program interruption element (PIE). Thus, a SPIE exit routine should save any required PIE data before issuing a SPIE. If a caller issues an ESPIE macro from within a SPIE exit routine, it has no effect on the contents of the PIE. However, if an ESPIE macro deletes the last SPIE/ESPIE environment, the PIE is freed and the SPIE exit cannot retry.

If the current SPIE environment is cancelled during SPIE exit routine processing, the control program will not return to the interrupted program when the SPIE program terminates. Therefore, if the SPIE exit routine wishes to retry within the interrupted program, a SPIE cancel should not be issued within the SPIE exit routine.

The SPIE macro can be issued by any problem program being executed in the performance of the task. The control program automatically deletes the SPIE exit routine when the request block (RB) that issued the SPIE macro terminates.

A PICA (program interruption control area) is created as part of the expansion of SPIE. The PICA contains the exit routine's address and a code indicating the interruption types specified in SPIE.

For more information on the SPIE macro, see the sections on program interruption services in *z/OS MVS Programming: Assembler Services Guide* and *z/OS MVS Programming: Authorized Assembler Services Guide*.

The following description of the SPIE macro also appears in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*, with the exception of interruption type 17. This interruption type designates page faults and its use is restricted to an authorized program.

SPIE Macro

Environment

The requirements for the caller are:

Minimum authorization:	To issue SPIE without encountering an abnormal end, callers must be in problem state, with a PSW key value that is equal to the TCB assigned key. To specify page fault processing, the caller must be APF-authorized.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in the primary address space

Programming Requirements

The caller must include the following mapping macros:

- IHAPIE
- IHAPICA

Restrictions

None.

Input Register Information

Before issuing the SPIE macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output Register Information

When control returns to the caller, the general purpose registers (GPRs) contain the following information:

Register	Contents
0	Used as a work register by the system.
1	If a SPIE environment is already active when you issue the SPIE macro, the SPIE service routine returns the address of the previous PICA in register 1. You can use this PICA to restore the previously active SPIE environment. However, if an ESPIE environment is active when you issue the SPIE macro, the SPIE service returns the address, in register 1, of a PICA in which the first word contains binary zeros. You cannot modify the contents of this PICA, and it contains no useful information except to restore the previous SPIE or ESPIE environment. If no previous SPIE/ESPIE environment is active, the service routine returns a zero in register 1.
2-13	Unchanged.
14-15	Used as a work register by the system.

When control returns to the caller, the access registers (ARs) contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance Implications

None.

Syntax

The standard form of the SPIE macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
<i>b</i>	One or more blanks must precede SPIE.
SPIE	
<i>b</i>	One or more blanks must follow SPIE.

<i>exit addr</i>	<i>exit addr</i> : A-type address, or register (2) - (12).
<i>,(interrupts)</i>	<i>interrupts</i> : Decimal numbers 1-15, or 17 expressed as single values : (2,3,4,7,8,9,10) ranges of values : ((2,4),(7,10)) combinations : (2,3,4,(7,10))

Parameters

The parameters are explained as follows:

exit addr

Specifies the address of the exit routine to be given control when a specific program interruption occurs. The exit routine receives control in 24-bit addressing mode.

,(interrupts)

Indicates the type of interruption for which the exit routine is to be given control. The interruption types are as follows:

Number	Interruption Type
1	Operation
2	Privileged operation
3	Execute
4	Protection
5	Addressing
6	Specification
7	Data
8	Fixed-point overflow (maskable)
9	Fixed-point divide
10	Decimal overflow (maskable)

SPIE Macro

11	Decimal divide
12	Exponent overflow
13	Exponent underflow (maskable)
14	Significance (maskable)
15	Floating-point divide
17	Page fault

Notes:

1. If an exit address is zero or no parameters are specified, the current SPIE and any previously active ESPIE environments are cancelled.
2. If a program interruption type is maskable, the corresponding program mask bit in the PSW (program status word) is set to 1 when specified and to 0 when not specified. Interruption types that are not maskable and not specified above are handled by the system, which forces an abend with the program check as the completion code. If an ESTAE-type recovery routine is also active, the SDWA indicates a system-forced abnormal termination. The registers at the time of the error are those of the system.
3. If you are using vector instructions and an interruption of 8, 12, 13, 14, or 15 occurs, your recovery routine can check the exception extension code (the first byte of the two-byte interruption code in the EPIE or PIE) to determine whether the exception was a vector or scalar type of exception.

ABEND Codes

The SPIE macro might return the following abend codes:

X'10E'
X'30E'
X'46D'.

See *z/OS MVS System Codes* for explanations and programmer responses.

Return and Reason Codes

None.

Example

Give control to an exit routine for interruption 17. DOITSPIE is the address of the SPIE exit routine.

SPIE DOITSPIE,(17)

SPIE—List Form

Use the list form of the SPIE macro to construct a control program parameter list in the form of a program interruption control area.

Syntax

The list form of the SPIE macro is written as follows:

name

name: Symbol. Begin *name* in column 1.

b

One or more blanks must precede SPIE.

SPIE

b One or more blanks must follow SPIE.

<i>exit addr</i>	<i>exit addr</i> : A-type address.
<i>, (interrupts)</i>	<i>interrupts</i> : Decimal numbers 1-15, or 17, expressed as
:	single values : (2,3,4,7,8,9,10)
:	ranges of values : ((2,4),(7,10))
:	combinations : (2,3,4,(7,10))
<i>,MF=L</i>	

Parameters

The parameters are explained under the standard form of the SPIE macro, with the following exception:

,MF=L

Specifies the list form of the SPIE macro.

SPIE—Execute Form

A remote control program parameter list is used in, and can be modified by, the execute form of the SPIE macro. The PICA (program interruptions control area) can be generated by the list form of SPIE, or you can use the address of the PICA returned in register 1 following a previous SPIE macro. If this macro is being issued to reestablish a previous SPIE environment, code only the MF parameter.

The address of the remote control program parameter list associated with any previous SPIE environment is returned by the SPIE macro.

Syntax

The execute form of the SPIE macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede SPIE.
SPIE	
b	One or more blanks must follow SPIE.

<i>exit addr</i>	<i>exit addr</i> : RX-type address, or register (2) - (12).
<i>, (interrupts)</i>	<i>interrupts</i> : Decimal numbers 1-15, or 17, expressed as

SPIE Macro

single values: (2,3,4,7,8,9,10)
ranges of values: ((2,4),(7,10))
combinations: (2,3,4,(7,10))

,MF=(E,*ctrl addr*)

ctrl addr: RX-type address, or register (1) or (2) - (12).

Parameters

The parameters are explained under the standard form of the SPIE macro, with the following exception:

,MF=(E,*ctrl,addr*)

Specifies the execute form of the SPIE macro using a remote control program parameter list.

Note: If SPIE is coded with a zero as the control address, the SPIE environment is canceled.

SPOST — Synchronize POST

Description

The SPOST macro is used in a cross memory post environment to ensure that all outstanding cross memory post requests to the current address space have completed. SPOST resolves a synchronization problem that arises when it becomes necessary to free an ECB that is a potential target for a cross memory post request. Before issuing SPOST, you must stop any new posts from being initiated.

For explanation of the parameters in a cross memory post request, see the POST macro.

Environment

These are the requirements for the caller:

Minimum authorization:	Supervisor state or PSW key 0-7 or APF authorized
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	None

Programming Requirements

The caller must include the CVT mapping macro.

Restrictions

None.

Input Register Information

Before issuing the SPOST macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter or using it as a base register.

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

When control returns to the caller, the ARs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

SPOST Macro

Performance Implications

None.

Syntax

The SPOST macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
␣	One or more blanks must precede SPOST.
SPOST	
␣	One or more blanks must follow SPOST.

Note: SPOST contains no optional or required parameters.

ABEND Codes

17B
27B
47B

See *z/OS MVS System Codes* for an explanation and programmer responses for this code.

Return and Reason Codes

None.

Example

Execute the SPOST macro with a comment.

```
SPOST          ,ISSUE SPOST
```

SRBSTAT — Save, Restore, or Modify SRB Status

Description

Note: IBM recommends that you use the following macros rather than SRBSTAT:

- SUSPEND - Suspend Execution of an SRB
- RESUME - Resume or Purge a Suspended SRB

The SRBSTAT macro allows the caller to save, restore, and modify the status of an SRB in a caller-supplied savearea. Control returns from the SRBSTAT macro in primary ASC mode.

Environment

These are the requirements for the caller:

Minimum authorization:	Supervisor state, key 0
Dispatchable unit mode:	SRB for SAVE or RESTORE options. SRB or TASK for MODIFY option.
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in primary address space

Programming Requirements

None.

Restrictions

The caller of SRBSTAT RESTORE must not change PASID before the request; the RESTORE will restore the PASID saved by the earlier SAVE request.

Input Register Information

Before issuing the SRBSTAT macro, the caller must ensure that general purpose register 13 points to a standard 72-byte save area addressable in primary mode.

Output Register Information

When control returns to the caller, the GPRs contain the following.

When SAVE is specified:

Register	Contents
0	Unchanged
1	Used as a work register by the system
2-14	Unchanged
15	Return code

When RESTORE is specified:

Register	Contents
0-13	Restored

SRBSTAT Macro

14	Unchanged
15	Return code

When MODIFY is specified:

Register	Contents
0-14	Unchanged
15	Return code

When control returns to the caller, the ARs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

Performance Implications

None.

Syntax

The SRBSTAT macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede SRBSTAT.
SRBSTAT	
b	One or more blanks must follow SRBSTAT.

SAVE	
RESTORE	
MODIFY	
 , <i>STSV=stsv addr</i>	 <i>stsv addr</i> . RX-type address or register (1) - (12), register (1) preferred.
 , <i>STSV=0</i>	
 , <i>NEWFRR=addr</i>	 <i>addr</i> . RX-type address or register (0) or (2) - (12), register (0) preferred.
 , <i>PRGAT=pat addr</i>	 <i>pat addr</i> . RX-type address or register (2) - (12), register (2) preferred.

Parameters

The parameters are explained as follows:

SAVE
RESTORE

MODIFY

Specifies whether a save, restore, or modify operation is requested. For SAVE or RESTORE, only the following status is saved or restored:

- General and floating point registers
- Control registers 3 and 4
- CPU affinity mask
- Related ASID/TCB
- Timing information
- FRR stack
- PCLINK stack header

If SAVE is specified, only caller's registers 1 and 15 are destroyed. Register 1 is used to hold an FRR parameter area address if NEWFRR is also specified and register 15 is used for a return code. The PCLINK stack header is saved and zeroed.

If RESTORE is specified, registers 0-13 are restored. The contents of register 14 are the same as when RESTORE was entered. The current PCLINK stack header must be zero; the saved one is restored.

If MODIFY is specified, registers 0-14 are unchanged and register 15 contains a return code.

Notes:

1. On entry to RESTORE, the PCLINK stack header must be zero.
2. RESTORE cannot be used in an FRR.
3. RESTORE returns to its caller and not to the caller of SAVE.
4. SRBSTAT does not save and restore access registers, extended authorization index (EAX) value, and linkage stack and access list status.

,STSV=*stsv addr*

Specifies the address of the savearea to be used for the SAVE, RESTORE, or MODIFY operation. The size of this savearea is contained in field SVTSSTSV of the SVT control block. The savearea can be in private pageable storage, but it must be addressable from the home address space and it must begin on a doubleword boundary. For RESTORE or MODIFY, the savearea must contain valid status.

,STSV=0

Specifies that the current status is to be modified. This parameter is valid only with MODIFY.

For MODIFY, an existing SRB status savearea or the current status is modified. Only the purge ASID/TCB information can be modified. All registers are saved and restored except register 15, which contains a return code.

Hexadecimal Code	Meaning
00	The modify function was successfully completed.

,NEWFRR=*addr*

Specifies the address of an FRR established with MODE=FULLXM. For SAVE, the address of the FRR parameter area is returned to the caller in register 1. The first word of the parameter area contains the address of the SRB status savearea being used.

SRBSTAT Macro

For RESTORE, the FRR address is used only if the saved status cannot be reinstated on the current processor. An SRB with the FRR option is scheduled specifying this FRR.

For MODIFY, this parameter is invalid.

,PRGAT=*pat addr*

Specifies the address of a 6-byte area of storage, currently addressable in the primary address space, that contains the new purge ASID/TCB. Bytes 1 and 2 contain the ASID; bytes 3-6 contain the TCB address. This parameter is required with MODIFY but is invalid with SAVE or RESTORE.

ABEND Codes

05E

See *z/OS MVS System Codes* for an explanation and programmer responses for this code.

Return and Reason Codes

None.

SRBTIMER — Establish Time Limit for System Service

Description

The SRBTIMER macro is used to establish a time limit for a system service running in SRB mode. Time accumulates while the service is running; when the time limit expires, the service abends with a system completion code of X'05B'. The service can retry following the 05B ABEND.

The caller can cancel an established time limit by reissuing the macro and specifying a time limit of zero. The caller can also override the established time limit with a subsequent SRBTIMER macro.

Environment

These are the requirements for the caller:

Minimum authorization:	Supervisor state and key 0
Dispatchable unit mode:	SRB
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No requirement
Control parameters:	Must be in primary address space

Programming Requirements

None.

Restrictions

None.

Input Register Information

Before issuing the SRBTIMER macro, the caller must ensure that general purpose register 13 points to a standard 72-byte save area addressable in primary mode.

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14	Used as a work register by the system
15	Return code

When control returns to the caller, the ARs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

SRBTIMER Macro

Performance Implications

None.

Syntax

The SRBTIMER macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
‡	One or more blanks must precede SRBTIMER.
SRBTIMER	
‡	One or more blanks must follow SRBTIMER.

LIMIT= <i>stor addr</i>	<i>stor addr</i> : RX-type address or register (0) or (2) - (12).
,ERRET= <i>err rtn addr</i>	<i>err rtn addr</i> : RX-type address or register (2) - (12).

Parameters

The parameters are explained as follows:

LIMIT=*stor addr*

Specifies the virtual storage address of a doubleword field on a doubleword boundary that contains the time limit. The time limit is in the form of a signed 64-bit binary number and must be positive in order for time to elapse. A negative number causes immediate expiration of the time limit. Bit 51 of the binary number is approximately equivalent to one microsecond. If you specify a value greater than 208 days, the control program changes the value to 208 days. The resolution of the timer is model dependent. See *Principles of Operation* for details concerning the timer facility.

,ERRET=*err rtn addr*

Specifies the address of the routine to be given control when the SRBTIMER function encounters damaged clocks.

ABEND Codes

None.

Return Codes

When SRBTIMER macro returns control to your program, GPR 15 contains a hexadecimal return code.

Table 15. Return Codes for the SRBTIMER Macro

Return Code	Meaning and Action
00	Meaning: The time limit was successfully established. Action: None.
10	Meaning: The issuer is not in SRB mode. No timing is performed. Action: Ensure that the requesting program is running in SRB mode, or use STIMER or STIMERM instead.

SRBTIMER Macro

STATUS — Stop, Start, or Put a Subtask in Process Must-Complete Mode

Description

The STATUS macro with the START or STOP option starts or stops a subtask. The STATUS macro with the SET or RESET option places a program in process-must-complete mode or ends process-must-complete mode. Process-must-complete mode postpones delays from the following:

- Asynchronous exits
- Status stops (by issuing the STATUS macro with the STOP option)
- Timer exits
- Dumping
- Swapping
- Attention exits

Process-must-complete mode prevents a CANCEL command from stopping a program already running but it does not postpone external interrupts or interrupts from I/O.

The description of the STATUS macro has two parts: the START/STOP option and the SET/RESET option. *z/OS MVS Programming: Assembler Services Reference ABE-HSP* describes the STATUS macro START and STOP options with the exception of the SRB and ASID parameters. These parameters, which are available only to supervisor state and key 0 callers, allow the caller to manipulate the dispatchability status or SRBs.

See *z/OS MVS Programming: Authorized Assembler Services Guide* for more information on how to use the STATUS macro SET and RESET options and *z/OS MVS Programming: Assembler Services Guide* for more information on how to use the STATUS macro START and STOP options.

Environment

The requirements for the caller issuing STATUS with the START or STOP parameters are:

Minimum authorization:	Problem state or supervisor state, with any PSW key. For SRB and ASID parameters, supervisor state and PSW key 0.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN or PASN¬=HASN¬=SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	No requirements

The requirements for the caller issuing STATUS with the SET or RESET parameters are:

Minimum authorization:	Supervisor state, with PSW key 0.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN or PASN¬=HASN¬=SASN
AMODE:	31-bit

STATUS Macro

ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	No requirements

Programming Requirements

None.

Restrictions

Except for the TCB, all input parameters to this macro can reside in storage above 16 megabytes.

While in process-must-complete mode, a task cannot:

- Issue STATUS STOP,SRB or STATUS START,SRB
- Request the LOCAL lock unconditionally
- Issue an SVC or invoke services that issue SVCs
- Issue the WAIT macro or invoke services that issue WAITs

These restrictions also apply to any routine invoked by a program in process-must-complete mode.

Process-must-complete mode is not preserved across a retry from an ESTAE-type routine. However, it is preserved across a retry from an FRR.

The caller cannot have an EUT FRR established.

Register Information

After the caller issues the macro, the system might use some registers as work registers or might change the contents of some registers. When the system returns control to the caller, the contents of these registers are not the same as they were before the macro was issued. Therefore, if the caller depends on these registers containing the same value before and after issuing the macro, the caller must save these registers before issuing the macro and restore them after the system returns control.

When control returns to the caller, the GPRs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14	Used as a work register by the system
15	Return code

When control returns to the caller, the access registers (ARs) contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

Performance Implications

Using STATUS will degrade performance of the calling program's address space while STATUS runs.

Remaining in process-must-complete mode for an extended period of time will degrade the performance of other programs waiting to use global resources that the program in this mode holds.

START/STOP Options

Syntax

The START/STOP options of the STATUS macro are written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede STATUS.
STATUS	
b	One or more blanks must follow STATUS.
START	
STOP	
,TCB= <i>tcb addr</i>	<i>tcb addr</i> : RX-type address or address in register (2) - (12), or 0.
,SRB	<i>ASID addr</i> : RX-type address or address in register (2) - (12).
,SRB, ASID= <i>ASID addr</i>	Note: ASID may only be specified with START.
,RELATED= <i>value</i>	<i>value</i> : Any valid macro keyword specification.

Parameters

The parameters are described as follows:

START STOP

Specifies that the task identified in the TCB parameter is to be stopped (STOP) or started (START). If you omit the TCB parameter, all subtasks of the originating task are stopped or started.

Note: The STOP parameter does not ensure that the subtask is stopped when control returns to the issuer. A subtask can have a “stop deferred” condition that would cause that particular subtask to remain dispatchable until stops are no longer deferred. In an MP environment, it would be possible to have a task issue the STATUS macro with the STOP parameter and resume processing while the subtask (for which the STOP was issued) is redispached to another processor.

,TCB=*tcb addr*
,SRB
,SRB,ASID=*ASID addr*

Specifies the status of the stop/start function:

STATUS Macro

TCB Specifies the address of a fullword on a fullword boundary containing the address of the TCB that is to have its START/STOP count adjusted.

Note: The TCB resides in storage below 16 megabytes.

SRB Specifies that the STOP and START functions affect the dispatchability of system-level SRBs; all tasks in the address space except the caller's are also set or reset nondispatchable. For START, the *ASID addr* specifies the address of a halfword containing the address space identifier. If ASID is not specified, the action is taken against the caller's address space.

,RELATED=*value*

Specifies information used to self-document macros by “relating” functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user and may be any valid coding values.

Return Codes

When control returns to the caller, register 15 contains one of the following hexadecimal return codes:

Table 16. Return Codes for the STATUS Macro

Return Code	Meaning and Action
00	Meaning: Processing completed successfully. Action: None required.
04	Meaning: Program error. START/STOP request failed. The task you specified is not a subtask of the calling program's task. Action: Ensure that the task you specify on the TCB parameter is a subtask of the calling program.

SET/RESET Option

Syntax

The SET/RESET option of the STATUS macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede STATUS.
STATUS	
b	One or more blanks must follow STATUS.

SET,MC,PROCESS
RESET,MC,PROCESS

Parameters

The parameters are explained as follows:

SET,MC,PROCESS

RESET,MC,PROCESS

SET,MC,PROCESS places the program that invokes the macro in process-must-complete mode.

RESET,MC,PROCESS ends process-must-complete mode.

Return Codes

When control returns to the caller, register 15 contains one of the following hexadecimal return codes:

Table 17. Return Codes for the SET/RESET Option

Return Code	Meaning and Action
00	Meaning: Processing completed successfully. Action: None required.
04	Meaning: Program error. You issued STATUS SET,MC,PROCESS but it had already been issued. Action: Do not issue STATUS SET,MC,PROCESS again until you have issued STATUS RESET,MC,PROCESS.
08	Meaning: Program error. You issued STATUS RESET,MC,PROCESS but had never issued STATUS SET,MC,PROCESS. Action: Issue STATUS SET,MC,PROCESS before issuing STATUS RESET,MC,PROCESS.

Example

Cause a program process to enter, then end, process-must-complete mode.

- * Chaining for a nonreenterable program, but note that STATUS SET,MC,
- * PROCESS and STATUS RESET,MC,PROCESS do not require that a savearea
- * be provided.

```

SETPMC  CSECT
SETPMC  AMODE 31
SETPMC  RMODE ANY
STM     14,12,12(13)      SAVE REGISTERS
LR      12,15             GET ENTRY POINT ADDRESS
USING   SETPMC,12         ESTABLISH ADDRESSABILITY
ST      13,SAVEAREA+4     SAVE REGISTER 13
LR      2,13              GET CALLER SAVEAREA ADDRESS
LA      13,SAVEAREA       SET UP OUR SAVEAREA ADDRESS
ST      13,8(2)           SAVE SAVEAREA ADDRESS IN CALLER
                               SAVEAREA
                               MODESET MODE=SUP,KEY=ZERO GET INTO SUPERVISOR STATE, KEY 0
STATUS  SET,MC,PROCESS    SET PMC MODE
LTR     15,15             CHECK RETURN CODE
BNZ     BADSET            NOT SUCCESSFUL, GO HANDLE...

*
* Perform processing that requires process-must-complete mode...
* Note: This processing must not request the local lock and
* must not issue any SVCs or issue a WAIT.
*
*
RESET   STATUS RESET,MC,PROCESS  RESET PMC MODE
LTR     15,15                   PMC MODE HAD ALREADY BEEN RESET,
```

STATUS Macro

```

                                ALREADY OUT OF PMC MODE
                                NOT SUCCESSFUL, GO HANDLE...
                                BNZ    BADRESET
*
EXIT    DS    0H
        MODESET MODE=PROB,KEY=NZERO GET OUT OF SUPERVISOR STATE, KEY 0
        L      13,4(13)             RESTORE REGISTER 13
        L      14,12(13)            RESTORE REGISTER 14
        LM     0,12,20(13)          RESTORE REGISTERS 0 THRU 12
        SLR    15,15                SET RETURN CODE 0 IN REGISTER 15
        BR     14                    RETURN TO THE CALLER
BADSET  DS    0H
*
*   Perform appropriate processing for nonzero return code from
*   STATUS SET,MC,PROCESS.
*
        B      EXIT
BADRESET DS    0H
*
*   Perform appropriate processing for nonzero return code from
*   STATUS RESET,MC,PROCESS.
*
        B      EXIT
SAVEAREA DC    18F'0'              18-WORD SAVEAREA
        END SETPMC

```

STORAGE — Obtain and Release Storage

Description

The STORAGE macro requests that the system obtain or release an area of virtual storage in the primary address space (by default), or in the address space defined through the ALET parameter. The two functions of the macro are:

- STORAGE OBTAIN, which requests one or more areas of virtual storage
- STORAGE RELEASE, which releases one or more areas of virtual storage.

The STORAGE macro is also described in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*, with the exception of the ALET, TCBADDR, and OWNER parameters.

If you use STORAGE OBTAIN to request real storage backing above 2 gigabytes, but your system does not support 64-bit storage, your request will be treated as a request for backing above 16 megabytes, even on earlier releases of OS/390 that do not support backing above 2 gigabytes. However, boundary requirements indicated by the CONTBDY and STARTBDY parameters will be ignored by earlier releases of OS/390.

Environment

The requirements for the caller are:

Minimum authorization:	For subpools 0-127: problem state and PSW key 8-15. For subpools 131 and 132, one or more of the following : <ul style="list-style-type: none">• Supervisor state• PSW key 0-7• PSW key mask (PKM) that allows the calling program to switch its PSW key to match the key of the storage to be obtained or released. For the ALET parameter, the TCBADDR parameter, and all other subpools, either of the following : <ul style="list-style-type: none">• Supervisor state• PSW key 0-7 To issue a subpool release for subpool 0: PSW key 0.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31- or 64-bit
ASC mode:	Primary or AR
Interrupt status:	Enabled for I/O and external interrupts. Enabled or disabled for I/O and external interrupts only if obtaining or releasing common storage.
Locks:	<ul style="list-style-type: none">• No requirement.• You may hold the local lock for the target address space.• If you hold the local lock, you may also hold the CMS lock.• You may hold the CPU lock when obtaining or releasing common storage.
Control parameters:	No requirement.

STORAGE Macro

Programming Requirements

None.

Restrictions

None.

Register Information

Register usage varies depending on the type of STORAGE request. For specific information, see the descriptions of STORAGE OBTAIN and STORAGE RELEASE.

Performance Implications

None.

STORAGE OBTAIN

The STORAGE macro with the OBTAIN parameter requests that the system allocate an area of virtual storage to the specified task. The length you specify must not exceed the length available. The length available depends on how much storage has already been allocated and, for subpools 0 - 127, 129-132, 240, and 250-252, the region size. For some subpools, the system releases the storage when the owning task terminates. For other subpools, you must issue STORAGE RELEASE or FREEMAIN to release them. Before obtaining storage, be sure to read "Selecting the Right Subpool for Your Virtual Storage Request" in *z/OS MVS Programming: Authorized Assembler Services Guide*.

Note: When you obtain storage, the system clears the requested storage to zeros if you obtain either:

- 8192 bytes or more from a pageable, private storage subpool.
- 4096 bytes or more from a pageable, private storage subpool, with BNDRY=PAGE specified.

The caller can specify CHECKZERO=YES to detect these and other cases where the system clears the requested storage to zeros.

Input Register Information

Before issuing the STORAGE macro with the OBTAIN parameter, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0	For a successful request in which maximum and minimum lengths were specified, contains the length of the storage obtained. Otherwise, used as a work register by the system.
1	The address of the allocated storage when STORAGE OBTAIN is successful; otherwise, used as a work register by the system.
	Note: In an AMODE 64 routine, a successful STORAGE OBTAIN will return a 64-bit pointer to the obtained area (bits 0-32 will be zero).
2-13	Unchanged
14	Used as a work register by the system.

15 Return code

When control returns to the caller, the access registers (ARs) contain:

Register	Contents
0	Used as a work register by the system.
1	The ALET value if you specified the ALET parameter and the STORAGE OBTAIN is successful. 0 if you did not specify the ALET parameter and the STORAGE OBTAIN is successful otherwise, used as a work register by the system..
2-13	Unchanged
14-15	Used as work registers by the system.

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the service returns control.

Syntax

The OBTAIN option of the STORAGE macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
␣	One or more blanks must precede STORAGE.
STORAGE	
␣	One or more blanks must follow STORAGE.

OBTAIN	
,LENGTH= <i>length value</i>	<i>length value</i> : Symbol, decimal number, or register (0), (2)-(12).
,LENGTH=(<i>max length,min length</i>)	<i>max length</i> : Symbol, decimal number, or register (0), (2) - (12). <i>min length</i> : Symbol, decimal number, or register (1) - (12).
,ADDR= <i>stor addr</i>	<i>stor addr</i> : RX-type address or register (1) - (12). Default: ADDR=(1).
,INADDR= <i>stor addr</i>	<i>stor addr</i> : RX-type address or register (1) - (12). Note: This parameter can only be specified with LOC=EXPLICIT.
,SP= <i>subpool number</i>	<i>subpool number</i> : Symbol, decimal number, or register (2)-(12), (15). Default: SP=0.
,ALET= <i>alet-value</i>	<i>alet-value</i> : Decimal number, RX-type address, or access register(1)-(12) Default: ALET=0.
,BNDRY=DBLWD	Default: BNDRY=DBLWD

STORAGE Macro

,BNDRY=PAGE	
,CONTBDY= <i>containing_bdy</i> ,STARTBDY= <i>starting_bdy</i>	<i>containing_bdy</i> : Decimal number 3-31 or register (2) - (12). <i>starting_bdy</i> : Decimal number 3-31 or register (2) - (12).
,KEY= <i>key number</i>	<i>key number</i> : Decimal number 0-15 or register (2) - (12). Note 1 : KEY is valid only when SP is specified. Note 2 : You cannot specify both KEY and CALLRKY=YES.
,CALLRKY=NO ,CALLRKY=YES	Default : CALLRKY=NO Note : You cannot specify both CALLRKY=YES and KEY.
,LOC=24 ,LOC=(24,31) ,LOC=(24,64) ,LOC=31 ,LOC=(31,31) ,LOC=(31,64) ,LOC=RES ,LOC=(RES,31) ,LOC=(RES,64) ,LOC=EXPLICIT ,LOC=(EXPLICIT,24) ,LOC=(EXPLICIT,31) ,LOC=(EXPLICIT,64)	Default : LOC=RES Note : You must specify the INADDR parameter with EXPLICIT.
,OWNER=HOME ,OWNER=PRIMARY ,OWNER=SECONDARY ,OWNER=SYSTEM	Default : OWNER=HOME
,RTCD= <i>rtcd addr</i>	<i>rtcd addr</i> : RX-type address or register (2) - (12) or (15). Default : RTCD=(15).
,COND=YES ,COND=NO	Default : COND=NO
,CHECKZERO=YES ,CHECKZERO=NO	Default : CHECKZERO=NO
,TCBADDR= <i>tcbaddress</i>	<i>tcbaddress</i> : RS-type address or register (2) - (12). Default : See "Selecting the Right Subpool for Your Virtual Storage Request" in <i>z/OS MVS Programming: Authorized Assembler Services Guide</i> for the possible default values.
,RELATED= <i>value</i>	<i>value</i> : Any valid macro parameter specification.

Parameters

The parameters are explained as follows:

OBTAIN

Requests that the system obtain virtual storage.

,LENGTH=*length value*

,LENGTH=(*max length,min length*)

Specifies the amount of storage the system is to obtain. *length value* specifies the length, in bytes, of the requested virtual storage. *max length* and *min length* specify the maximum and minimum amounts of storage. These numbers should be a multiple of 8; if they are not, the system uses the next higher multiple of 8.

,ADDR=*stor addr*

Specifies the location where the system is to return the address of the storage it allocates.

,INADDR=*stor addr*

Specifies the desired virtual address for the storage to be obtained. When you specify INADDR, you must specify EXPLICIT on the LOC parameter.

Notes:

1. The address specified on INADDR must be on a doubleword boundary.
2. Make sure that the virtual storage address specified on INADDR and the central storage backing specified on the LOC=EXPLICIT parameter are a valid combination. For example, if the address specified on INADDR is for storage above 16 megabytes, specify LOC=EXPLICIT or LOC=(EXPLICIT,ANY). Valid combinations include:
 - virtual above, central any
 - virtual any, central any
 - virtual below, central below
 - virtual below, central any

,SP=*subpool number*

Specifies the subpool number for the storage. (See *z/OS MVS Programming: Authorized Assembler Services Guide* for a list of valid subpools.) If you specify a register, the subpool number must be in bits 24-31 of the register, with bits 0-23 set to zero. If you omit this parameter, the system uses the default, which is subpool 0.

Notes:

1. Callers executing in supervisor state and key 0, who specify subpool 0, will obtain storage from subpool 252. Therefore, when requesting a dump of this storage through the SDUMP or SDUMPX macro, they must specify subpool 252 rather than zero.
2. Storage requested from subpools 240 and 250 are translated to subpool 0 requests.

,ALET=*alet-value*

Specifies the ALET of the target address space — the address space in which the storage is to be obtained. The ALET must be on the caller's primary address space access list (PASN-AL) or dispatchable unit access list (DU-AL) and, if the ALET identifies a private entry, the caller must be authorized to the target address space through the extended authorization index (EAX). For more information, see *z/OS MVS Programming: Extended Addressability Guide*. If you omit this parameter, the system assumes the target address space is in the primary address space.

,BNDRY=DBLWD

,BNDRY=PAGE

Specifies that alignment on a doubleword boundary (DBLWD) or alignment with the start of a virtual page on a 4K boundary (PAGE) is required for the start of a requested area.

STORAGE Macro

If the request specifies one of the LSQA or SQA subpools, the system ignores the BNDRY=PAGE keyword. Requests for storage from these subpools are then fulfilled from a single page, unless the request is greater than a page. See *z/OS MVS Programming: Authorized Assembler Services Guide* for a list of the LSQA and SQA subpools.

The default is BNDRY=DBLWD.

,CONTDY=*containing_bdy*

Specifies the boundary the obtained storage must be contained within. Specify a power of 2 that represents the containing boundary. Supported values are 3-31. For example, CONTDY=10 means the containing boundary is 2^{10} , or 1024 bytes. The containing boundary must be at least as large as the maximum requested boundary. The obtained storage will not cross an address that is a multiple of the requested boundary.

If a register is specified, the value must be in bits 24-31 of the register. Do not specify CONTDY on a variable-length request.

CONTDY is not valid with LOC=EXPLICIT or BNDRY=PAGE.

CONTDY applies to all subpools.

If you omit this parameter, there is no containing boundary.

,STARTBDY=*starting_bdy*

Specifies the boundary the obtained storage must start on. Specify a power of 2 that represents the start boundary. Supported values are 3-31. For example, STARTBDY=10 means the start boundary is 2^{10} , or 1024 bytes. The obtained storage will begin on an address that is a multiple of the requested boundary.

If a register is specified, the value must be in bits 24-31 of the register. Do not specify STARTBDY on a variable-length request.

STARTBDY is not valid with LOC=EXPLICIT or BNDRY=PAGE.

STARTBDY applies to all subpools.

If you omit this parameter, the start boundary is 8 bytes (equivalent to specifying STARTBDY=3).

,KEY=*key-number*

Indicates the storage key of the storage to be obtained. The valid storage keys are 0-15. If you pass the storage key in a register, it must be in bits 24-27 in that register. KEY is valid only when SP is specified, and applies only to subpools 129-132, 227-231, 241, and 249. The system ignores the KEY parameter if KEY is used for any other subpools. For detailed information about how the system determines what storage key to assign to your storage request, see "Selecting the Right Subpool for Your Virtual Storage Request" in the *z/OS MVS Programming: Authorized Assembler Services Guide*.

,CALLRKY=NO

,CALLRKY=YES

Specifies how the system assigns the key for the storage to be obtained:

CALLRKY=NO

The system assigns the value according to the specified subpool:

- For subpools 129-132, 227-231, 241, and 249, the system assigns the value specified on the KEY parameter (or zero, if the KEY parameter is omitted) as the storage key

- For all other subpools, the system ignores the CALLRKY parameter.

CALLRKY=YES

The system assigns the caller's current PSW key as the storage key. When you specify CALLRKY=YES, do not also specify KEY. Specify CALLRKY only when obtaining storage from subpools 129-132, 227-231, 241, and 249. For all other subpools, the system ignores the CALLRKY parameter.

The default is CALLRKY=NO. For detailed information about how the system determines what storage key to assign to your storage request, see "Selecting the Right Subpool for Your Virtual Storage Request" in the *z/OS MVS Programming: Authorized Assembler Services Guide*.

,LOC=24
 ,LOC=(24,31)
 ,LOC=(24,64)
 ,LOC=31
 ,LOC=(31,31)
 ,LOC=(31,64)
 ,LOC=RES
 ,LOC=(RES,31)
 ,LOC=(RES,64)
 ,LOC=EXPLICIT
 ,LOC=(EXPLICIT,24)
 ,LOC=(EXPLICIT,31)
 ,LOC=(EXPLICIT,64)

Specifies the location of virtual storage and central (also called real) storage. This is especially helpful for callers with 24-bit dependencies. When LOC is specified, central storage is allocated anywhere until the storage is fixed (for example, using the PGSER macro). You can specify the location of central storage (after the storage is fixed) and virtual storage (whether or not the storage is fixed) using the following LOC parameter values:

LOC=24 indicates that central and virtual storage are to be located below 16 megabytes. LOC=24 must not be used to allocate disabled reference (DREF) storage.

Note: Specifying LOC=BELOW is the same as specifying LOC=24. LOC=BELOW is still supported, but IBM recommends using LOC=24 instead.

LOC=(24,31) indicates that virtual storage is to be located below 16 megabytes and central storage can be located anywhere below 2 gigabytes.

Note: Specifying LOC=(BELOW,ANY) is the same as specifying LOC=(24,31). LOC=(BELOW,ANY) is still supported, but IBM recommends using LOC=(24,31) instead.

LOC=(24,64) indicates that virtual storage is to be located below 16 megabytes and central storage can be located anywhere in 64-bit storage.

LOC=31 and LOC=(31,31) indicate that virtual and central storage can be located anywhere below 2 gigabytes.

STORAGE Macro

Note: Specifying LOC=ANY or LOC=(ANY,ANY) is the same as specifying LOC=31 or LOC=(31,31). LOC=ANY and LOC=(ANY,ANY) are still supported, but IBM recommends using LOC=31 or LOC=(31,31) instead.

LOC=(31,64) indicates that virtual storage is to be located below 2 gigabytes and central storage can be located anywhere in 64-bit storage.

When you use LOC=RES to allocate storage that can reside either above or below 16 megabytes, LOC=RES indicates that the location of virtual and central storage depends on the location of the caller. If the caller resides below 16 megabytes, virtual and central storage are to be located below 16 megabytes. If the caller resides above 16 megabytes, virtual and central storage are to be located either above or below 16 megabytes.

LOC=(RES,31) indicates that the location of virtual storage depends upon the location of the caller. If the caller resides below 16 megabytes, virtual storage is to be located below 16 megabytes; if the caller resides above 16 megabytes, virtual storage can be located anywhere below 2 gigabytes. In either case, central storage can be located anywhere below 2 gigabytes.

Note: Specifying LOC=(RES,ANY) is the same as specifying LOC=(RES,31). LOC=(RES,ANY) is still supported, but IBM recommends using LOC=(RES,31) instead.

LOC=(RES,64) indicates that the location of virtual storage depends upon the location of the caller. If the caller resides below 16 megabytes, virtual storage is to be located below 16 megabytes; if the caller resides above 16 megabytes, virtual storage can be located anywhere in 31-bit storage. In either case, central storage can be located anywhere in 64-bit storage.

Note: If your program resides below 16 megabytes but runs with 31-bit addressing mode, you can specify LOC=RES (as a default or explicitly) or LOC=(RES,31) to obtain storage from a subpool supported only above 16 megabytes. Do not specify subpools supported only above 16 megabytes on requests using LOC=RES or LOC=(RES,31) if your program resides below 16 megabytes and runs with 24-bit addressing.

LOC=EXPLICIT, LOC=(EXPLICIT,24), LOC=(EXPLICIT,31), or LOC=(EXPLICIT,64) specify that the requested virtual storage is to be located at the address specified with the INADDR parameter, which is required with EXPLICIT. EXPLICIT is valid only for subpools 0-127, 129-132, 240, 244, 250, 251, and 252. You cannot specify the BNDRY, OWNER, or LENGTH=(*max length,min length*) parameters with EXPLICIT.

Note: Specifying LOC=(EXPLICIT,BELOW) is the same as specifying LOC=(EXPLICIT,24). Specifying LOC=(EXPLICIT,ANY) is the same as specifying LOC=(EXPLICIT,31). The older specifications are still supported, but IBM recommends using the newer specifications instead.

LOC=(EXPLICIT,31) indicates that virtual storage is to be located at the address specified on the INADDR parameter, and central storage can be located anywhere below 2 gigabytes.

LOC=(EXPLICIT,24) indicates that virtual storage is to be located at the address specified on the INADDR parameter, and central storage is to be located below 16 megabytes. The virtual storage address specified on the INADDR parameter must be below 16 megabytes.

LOC=EXPLICIT and LOC=(EXPLICIT,64) indicate that virtual storage is to be located at the address specified on the INADDR parameter, and central storage can be located anywhere in 64-bit storage.

When you specify EXPLICIT on a request for storage from the same virtual page as previously requested storage, you must request it in the same key, subpool, and central storage area as on the previous storage request. For example, if you request virtual storage backed with central storage below 16 megabytes, any subsequent requests for storage from that virtual page must be specified as LOC=(EXPLICIT,24).

Notes:

1. A caller cannot specify LOC=24 or LOC=(24,31) from subpools: 203-205, 213-215, 223-225, 247, and 248 because they are supported only above 16 megabytes.
2. When you specify LOC=31, the actual location of the virtual storage (that is, whether it is above or below 16Mb) depends on the subpool you specify on the SP parameter:
 - Some subpools (for example, subpool 226) are supported **only below** 16 megabytes. For these subpools, STORAGE OBTAIN locates virtual storage below 16 megabytes, regardless of how you specify LOC.
 - Some subpools (for example, 203-204) are supported **only above** 16 megabytes. For these subpools, STORAGE OBTAIN locates virtual storage above 16 megabytes. If you specify LOC=24 for one of these subpools, the system abends your program.
 - All other subpools are supported both above and below 16Mb. For these subpools, specifying LOC=31 causes STORAGE OBTAIN to try to allocate virtual storage above 16Mb, but below 2Gb. If the attempt fails, it tries to allocate virtual storage below 16Mb. If this attempt also fails, it does not allocate any storage.
3. A caller residing below 16 megabytes but running in 31-bit addressing mode can specify LOC=RES (as a default or explicitly) or LOC=(RES,31) to obtain storage from a subpool supported only above 16 megabytes.

,OWNER=HOME

,OWNER=PRIMARY

,OWNER=SECONDARY

,OWNER=SYSTEM

Specifies the entity to which the system will assign ownership of requested CSA, ECSA, SQA, and ESQA storage. The system uses this ownership information to track the use of CSA, ECSA, SQA and ESQA storage. This parameter can have one of the following values:

HOME The home address space

PRIMARY The primary address space

SECONDARY The secondary address space

SYSTEM The system (the storage is not associated with an address)

STORAGE Macro

space); specify this value if you expect the requested storage to remain allocated after termination of the job that obtained the storage.

The default value is OWNER=HOME. The system ignores the OWNER keyword unless you specify a CSA, ECSA, SQA, or ESQA subpool on the SP parameter.

Storage tracking is available as of MVS/SP release 4.3. However, programs that issue the STORAGE OBTAIN macro with the OWNER parameter can run on any MVS system from MVS/SP 3.1 to the current release.

,RTCD=rtcd addr

Specifies the location where the system is to store the return code. The return code is also in GPR 15. This parameter is valid only if you specify COND=YES.

,COND=YES

,COND=NO

Specifies whether the request is unconditional or conditional.

COND=YES specifies that the active unit of work should not be abnormally terminated if there is insufficient contiguous virtual storage to satisfy the request, and instead should return to the caller with a non-zero return code. Use of COND=YES does not prevent all abnormal terminations. For example, if the request has incorrect or inconsistent parameters, the system abnormally terminates the active unit of work. If you specify COND=YES, you may also specify the RTCD parameter to define the location where the system is to store the return code.

COND=NO indicates that the request is unconditional. The system abnormally terminates the active unit of work if the STORAGE OBTAIN request cannot complete successfully. This situation occurs if the parameters passed on the request are incorrect or inconsistent, if the system encounters internal errors, or if there is not enough contiguous virtual storage to satisfy the request.

COND=NO is the default.

,CHECKZERO=YES

,CHECKZERO=NO

Specifies whether or not the return code for a successful completion should indicate if the system has cleared the requested storage to zeroes. When CHECKZERO=NO is specified or defaulted, the return code for a successful completion is 0. When CHECKZERO=YES is specified, the return code for a successful completion is X'14' if the system has cleared the requested storage to zeroes, and 0 if the system has not cleared the requested storage to zeroes.

There is no performance cost to specifying CHECKZERO=YES.

CHECKZERO processing is available as of OS/390 R6. Programs that issue the STORAGE macro with the CHECKZERO parameter can run on any MVS system from MVS/SP 2.1 to the current release. On a down-level system, CHECKZERO will be ignored, and the return code for a successful completion (conditional or unconditional) will be 0.

,TCBADDR=tcbaddress

Specifies the address of a word that contains the address of the input task control block (TCB), or a register that contains the address of the input TCB. The system assumes that the input TCB resides in the address space where the storage is to be obtained.

For an explanation of the term **input TCB**, and to determine the system-assigned defaults for private storage ownership, see “Selecting the Right Subpool for Your Virtual Storage Request” in *z/OS MVS Programming: Authorized Assembler Services Guide*.

The system ignores the TCBADDR keyword if the STORAGE OBTAIN request is for a common storage subpool.

,RELATED=value

Specifies information used to self-document macro by “relating” functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user, and can be any valid coding values.

ABEND Codes

Abend codes that STORAGE OBTAIN might issue are listed below. For detailed abend code information, see *z/OS MVS System Codes*.

178	278	378	478	778
878	978	A78	B78	D78

Return and Reason Codes

When control returns from the STORAGE OBTAIN request and you specified a conditional request, GPR 15 (and *rtcd addr*, if you coded RTCD) contains one of the following hexadecimal return codes:

Table 18. Return Codes for STORAGE OBTAIN

Return Code	Meaning and Action
0	<p>Meaning: Successful completion. CHECKZERO=YES was not specified, or the system has not cleared the requested storage to zeroes.</p> <p>Action: None.</p>
4	<p>If you did not specify EXPLICIT on the LOC parameter:</p> <p>Meaning: Environmental or system error. Virtual storage was not obtained because insufficient storage is available.</p> <p>Action: If the request was for private (local) storage, consult the system programmer to see if you have exceeded an installation-determined private storage limit.</p> <p>If the request is for common (global) storage, your system is probably experiencing a common storage shortage and your request cannot be satisfied until the shortage is corrected.</p> <p>If you specified EXPLICIT on the LOC parameter:</p> <p>Meaning: Program error. Virtual storage was not obtained because part of the requested storage area is outside the bounds of the user region.</p> <p>Action: Determine why your program is mistakenly requesting storage outside the user region. If your region size is too small, consult the system programmer about increasing the region size.</p>
8	<p>Meaning: System error. Virtual storage was not obtained because the system has insufficient central storage to back the request.</p> <p>Action: Report the problem to the system programmer so the cause of the problem can be determined and corrected.</p>

STORAGE Macro

Table 18. Return Codes for STORAGE OBTAIN (continued)

Return Code	Meaning and Action
0C	<p>Meaning: System error. Virtual storage was not obtained because the system cannot page in the page table associated with the storage to be allocated.</p> <p>Action: Report the problem to the system programmer so the cause of the problem can be determined and corrected.</p>
10	<p>Meaning: Program error. Virtual storage was not obtained due to one of the reasons listed below. This return code applies only to STORAGE requests with LOC=EXPLICIT specified.</p> <ul style="list-style-type: none">• Part of the requested area is allocated already.• Virtual storage was already allocated in the same page as this request, but one of the following characteristics of the storage was different:<ul style="list-style-type: none">– The subpool– The key– Central storage backing <p>Action: Determine why your program is attempting to obtain allocated storage or why your program is attempting to obtain virtual storage with different attributes from the same page of storage. Correct the coding error.</p>
14	<p>Meaning: Successful completion. The system has cleared the requested storage to zeroes.</p> <p>This return code occurs only when CHECKZERO=YES is specified.</p> <p>Action: None.</p>

Examples

For examples of how to use the STORAGE macro with the OBTAIN option, see “Examples of the OBTAIN and RELEASE Options” on page 125.

STORAGE RELEASE

The STORAGE macro with the RELEASE parameter requests that the system release an area of virtual storage or an entire virtual storage subpool, previously allocated through the STORAGE or GETMAIN macro. The system abends the active task if the specified virtual storage does not start on a doubleword boundary or, for an unconditional request, if the specified area or subpool is not allocated to the current task. The current task is determined from the input task specified on the TCBADDR parameter (see “Selecting the Right Subpool for Your Virtual Storage Request” in *z/OS MVS Programming: Authorized Assembler Services Guide* for more information about the input task).

Input Register Information

Before issuing the STORAGE macro with the RELEASE parameter, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14	Used as a work register by the system

15 Return code if you specified COND=YES; otherwise, used as a work register by the system.

When control returns to the caller, the access registers (ARs) contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system.

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the service returns control.

Syntax

The RELEASE option of the STORAGE macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
␣	One or more blanks must precede STORAGE.
STORAGE	
␣	One or more blanks must follow STORAGE.

RELEASE	
,LENGTH= <i>length value</i>	<i>length value</i> : Symbol, decimal number, or register (0), (2) - (12).
,ADDR= <i>stor addr</i>	<i>stor addr</i> : RX-type address or register (1) - (12).
,LENGTH= <i>length value</i>	
,SP= <i>subpool number</i>	<i>subpool number</i> : Symbol, decimal number, or register (2) - (12), (15). Default: SP=0.
,ALET= <i>alet-value</i>	<i>alet-value</i> : Decimal number, RX-type address, access register (1) - (12). Default: ALET=0.
,KEY= <i>key number</i>	<i>key number</i> : Decimal number 0-15 or register (2) - (12). Note: KEY is valid only when SP is specified.
,CALLRKY=NO	Default: CALLRKY=NO
,CALLRKY=YES	Note: You cannot specify both CALLRKY=YES and KEY.
,RTCD= <i>rtcd addr</i>	<i>rtcd addr</i> : RX-type address or registers (2) - (12), (15). Default: RTCD=(15)
,COND=YES	Default: COND=NO
,COND=NO	
,TCBADDR= <i>tcbaddress</i>	<i>tcbaddress</i> : RS-type address or register (2) - (12).

STORAGE Macro

Default: See “Selecting the Right Subpool for Your Virtual Storage Request” in *z/OS MVS Programming: Authorized Assembler Services Guide* for the possible default values.

,RELATED=*value*

value: Any valid macro parameter specification.

Parameters

The parameters are explained as follows:

RELEASE

Requests that the system release virtual storage.

,LENGTH=*length value*

Specifies the number of bytes of storage that the system is to release. If you specify LENGTH, you must also specify ADDR. To free an entire subpool, use SP instead of LENGTH and ADDR. Do not specify a length value of zero with an address of zero. This will cause STORAGE RELEASE to free the subpool specified with the SP parameter, or subpool 0, if the SP parameter is omitted.

,ADDR=*stor addr*

Specifies the address of the storage to be released. If you specify ADDR, you must also specify LENGTH. To free an entire subpool, use SP instead of LENGTH and ADDR.

,SP=*subpool number*

Specifies the subpool number for the storage to be released. The subpool number must be a valid subpool number between 0 and 255. If you specify the subpool in a register, the subpool number must be in bits 24-31 of the register, with bits 0-23 set to zero. If you omit this parameter, the system uses subpool 0.

A request to release all the storage in a subpool is known as a **subpool release**. To issue a subpool release, use SP to indicate the subpool and do not specify either LENGTH or ADDR. Issue subpool releases only for the following subpools: 0-127, 129-132, 203, 204, 213, 214, 223, 224, 229, 230, 233, 236, 237, 240, 249, and 250-253. If you try to issue a subpool release for any other subpool, an abend X'478' or X'40A' occurs. See the list of subpool characteristics in *z/OS MVS Programming: Authorized Assembler Services Guide* for information and requirements pertaining to specific subpools.

Notes:

1. The system translates subpool 0 storage requests to subpool 252 storage requests when you are running in supervisor state and key 0. If you are not running in supervisor state and key 0, you will receive storage from subpool 0 when you request it.
2. The system translates subpool 240 and 250 storage requests to subpool 0 storage requests. Bearing this in mind, you must be careful to specify the correct subpool when obtaining and releasing storage. For instance, if you obtain subpool 0 storage while running in problem state, you will receive subpool 0 storage. If you attempt to release it after switching to supervisor state and PSW key 0, you cannot specify subpool 0 because the system will try to free subpool 252 storage. Instead, you must release the storage specifying subpools 240 or 250, which are translated by the system to subpool 0.

,ALET=alet-value

Specifies the ALET of the address space in which the storage is to be released. The ALET must be on the caller's primary address space access list (PASN-AL) or dispatchable unit access list (DU-AL) and, if the ALET identifies a private entry, the caller must be authorized to the target address space through the extended authorization index (EAX). For additional information, see *z/OS MVS Programming: Extended Addressability Guide*. If you omit this parameter, the system assumes storage is in the primary address space.

,KEY=key-number

Indicates the storage key of the storage to be released. The valid storage keys are 0-15. If you pass the storage key in a register, it must be in bits 24-27 in that register. KEY is valid only with SP and applies only to subpools 129-132, 227-231, 241, and 249. The system ignores the KEY parameter if KEY is used for any other subpools. KEY allows you to release storage in the specified storage protection key. See list of subpool characteristics in *z/OS MVS Programming: Authorized Assembler Services Guide* for information on authorization requirements pertaining to specific subpools.

,CALLRKEY=NO
,CALLRKEY=YES

Specifies how the system assigns the key for the storage to be obtained:

CALLRKY=NO

The system assigns the value according to the specified subpool:

- For subpools 129-132, 227-231, 241, and 249, the system assigns the value specified on the KEY parameter (or zero, if the KEY parameter is omitted) as the storage key
- For all other subpools, the system ignores the CALLRKY parameter.

CALLRKY=YES

The system assigns the caller's current PSW key as the storage key. When you specify CALLRKY=YES, do not also specify KEY. Specify CALLRKY only when obtaining storage from subpools 129-132, 227-231, 241, and 249. For all other subpools, the system ignores the CALLRKY parameter.

The default is CALLRKY=NO. For detailed information about how the system determines what storage key to assign to your storage request, see "Selecting the Right Subpool for Your Virtual Storage Request" in the *z/OS MVS Programming: Authorized Assembler Services Guide*.

,RTCD=rtcd addr

Specifies the location where the system is to store the return code. The return code is also in GPR 15. This parameter is only valid if you specify COND=YES.

,COND=YES
,COND=NO

Specifies whether the request is unconditional or conditional.

COND=YES specifies that the task should not abend if the system cannot release the storage. However, the system cannot prevent some abends. The RTCD parameter specifies the location where the system is to store a return code. COND=NO specifies that the system abend the active task if it cannot release the storage.

COND=NO is the default.

STORAGE Macro

,TCBADDR=*tcbaddress*

Specifies the address of a word that contains the address of the input task control block (TCB), or a register that contains the address of a word that contains the address of the input TCB.

For an explanation of the term **input TCB**, and to determine the system-assigned defaults for private storage ownership, see “Selecting the Right Subpool for Your Virtual Storage Request” in *z/OS MVS Programming: Authorized Assembler Services Guide*.

The system ignores the TCBADDR keyword if the STORAGE RELEASE request is for a common storage subpool. If you specified TCBADDR on STORAGE OBTAIN, you should also specify TCBADDR on STORAGE RELEASE.

,RELATED=*value*

Specifies information used to self-document macro by “relating” functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user, and can be any valid coding values.

ABEND Codes

Abend codes that STORAGE RELEASE might issue are listed below. For detailed abend code information, see *z/OS MVS System Codes*.

178	278	378	478	778
878	978	A78	B78	D78

Return and Reason Codes

When the STORAGE macro returns control to your program and you specified a conditional request, GPR 15 (and *rtcd addr*, if you coded RTCD) contains one of the following hexadecimal return codes:

Table 19. Return Codes for STORAGE RELEASE

Return Code	Meaning and Action
0	Meaning: Successful completion. Action: None.
4	Meaning: Program error. Not all requested virtual storage was freed. Action: Check your program for the following kinds of errors: <ul style="list-style-type: none">• The address of the storage area to be freed is not correct.• The subpool you have specified does not match the subpool of the storage to be freed.• The key you have specified does not match the key of the storage to be freed.• For private storage: the owning task identified by the input TCB is not correct for the storage to be freed.
8	Meaning: Program error. No virtual storage was freed because part of the storage area to be freed is fixed. Action: Check your program for the following kinds of errors: <ul style="list-style-type: none">• You passed an incorrect storage area address to the STORAGE macro.• You attempted to free storage that is fixed.

Examples of the OBTAIN and RELEASE Options

Example 1

Code the instructions to obtain 1000 bytes of virtual storage from subpool 223. The system returns the address of the storage in register 3. If the request fails, the system abends the caller.

```
LA      2,1000
STORAGE OBTAIN,LENGTH=(2),ADDR=(3),SP=223,COND=NO,LOC=ANY
```

Release the 1000 bytes obtained above from subpool 223 and abend the caller if the request fails. Assume that the length of the storage is still in register 2 and the address of the storage is in register 3.

```
STORAGE RELEASE,LENGTH=(2),ADDR=(3),SP=223,COND=NO
```

```
.
.
```

Example 2

Code the instructions to obtain 4096 bytes of virtual storage from subpool 227 — above 16 megabytes, if possible. The address is returned at location STRGA. The protection key is the caller's PSW key. The system is to assign the storage to be obtained to the primary address space. The system is to store the return code at location MY_RC.

```
STORAGE OBTAIN,LENGTH=ONE_PAGE,ADDR=STRGA,SP=MY_SUBPOOL,
        CALLRKY=YES,LOC=ANY,COND=YES,OWNER=PRIMARY,RTCD=MY_RC
```

To release the 4096 bytes obtained above from subpool 227, issue:

```
L      2,KEY_5
STORAGE RELEASE,LENGTH=ONE_PAGE,ADDR=STRGA,SP=MY_SUBPOOL,
        KEY=(2),COND=YES,RTCD=MY_RC
```

```
.
.
```

```
MY_RC   DS    F
STRGA   DS    F
KEY_5   DC    X'00000050'
ONE_PAGE EQU 4096
MY_SUBPOOL EQU 227
```

Note that, when the caller passes the key in a register, the key must be in bits 24-27. Note also, that KEY=KEY_5 is not valid, as KEY_5 is neither a register nor a decimal number.

Example 3

Code the instructions to obtain 4096 bytes of virtual storage from subpool 227. Indicate that, if the system cannot obtain 4096 bytes, the caller can settle for as little as 1024 bytes. The system returns the address of the storage obtained at location STRGA. The protection key is 5. The system is to store the return code at location MY_RC.

```
STORAGE OBTAIN,LENGTH=(ONE_PAGE,ONE_K),ADDR=STRGA,                                X
        SP=MY_SUBPOOL,KEY=5,LOC=ANY,COND=YES,RTCD=MY_RC
ST      0,STRG_LEN
```

Release the storage from subpool 227, obtained above. Note that you cannot specify LENGTH=STRG_LEN.

```
L      2,KEY_5
L      3,STRG_LEN
STORAGE RELEASE,LENGTH=(3),ADDR=STRGA,SP=MY_SUBPOOL,                                X
        KEY=(2),COND=YES,RTCD=MY_RC
```

```
.
.
```

STORAGE Macro

```
STRG_LEN    DS    F
MY_RC       DS    F
STRGA       DS    F
KEY_5       DC    X'00000050'
ONE_K       EQU   1024
ONE_PAGE    EQU   4096
MY_SUBPOOL  EQU   227
```

Example 4

Code the instructions to set up an 18-word save area, such as one that a program in AR address space control (ASC) mode would obtain to call a program in primary mode. The program issuing the STORAGE macro is in 31-bit addressing mode, and the code is reentrant.

```
PGM    CSECT
PGM    AMODE 31
PGM    RMODE ANY
      BAKR    14,0          SAVE CALLER'S ARS, GPRS AND RETURN
*                                ADDRESS ON LINKAGE STACK
      SAC     512           SWITCH TO AR ASC MODE
      LAE     12,0(15,0)    SET UP PROGRAM BASE REGISTER AND AR
      USING   PGM,12
      STORAGE OBTAIN,LENGTH=72 GET REENTRANT SAVEAREA
      LAE     13,0(1,0)     PUT SAVEAREA ADDRESS IN AR/GPR 13
      MVC     4(4,13),=C'F1SA' PUT ACRONYM INTO SAVEAREA TO
*                                INDICATE STATUS SAVED ON LINKAGE STACK
      .
*    BEGIN PROGRAM CODE HERE
```

To release this save area, issue the following instructions:

```
      .
      LAE     1,0(0,13)     COPY SAVEAREA ADDRESS
      STORAGE RELEASE,ADDR=(1),LENGTH=72 FREE SAVEAREA
      .
      SLR     15,15         SET RETURN CODE OF ZERO
      PR                                RETURN TO CALLER, RESTORE CALLERS STATUS
```

SUSPEND — Suspend Execution of an RB

Description

Suspend an SRB

To suspend an SRB, use the variation of the SUSPEND macro described under “SUSPEND — Suspend Execution of an SRB” on page 129.

To suspend execution of a request block (RB), use this variation of the SUSPEND macro. The RB remains suspended until a subsequent RESUME macro causes the RB to resume execution.

Environment

The requirements for the caller are:

Minimum authorization:	Supervisor state with PSW key 0
Dispatchable unit mode:	Task
Cross memory mode:	Any
AMODE:	24- or 31-bit
ASC mode:	Primary or secondary
Interrupt status:	Can be either enabled or disabled
Locks:	Can hold the CPU or local lock
Control parameters:	Must be in the caller's primary address space

Programming Requirements

The caller must include the IHAPSA mapping macro and the CVT mapping macro specifying DSECT=YES.

Restrictions

The list and execute forms of the SUSPEND macro are not valid for suspending execution of an RB.

Input Register Information

Before issuing the SUSPEND macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter or using it as a base register.

Output Register Information

After the caller issues the macro, the system might use some registers as work registers or might change the contents of some registers. When the system returns control to the caller, the contents of these registers are not the same as they were before the macro was issued. Therefore, if the caller depends on these registers containing the same value before and after issuing the macro, the caller must save these registers before issuing the macro and restore them after the system returns control.

When control returns to the caller, the general purpose registers (GPRs) contain:

Register	Contents
0	Address of the suspended TCB
1	Address of the suspended RB

SUSPEND Macro for RBs

2-10	Unchanged
11-15	Used as work registers by the system

Performance Implications

None.

Syntax

The SUSPEND macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
␣	One or more blanks must precede SUSPEND.
SUSPEND	
␣	One or more blanks must follow SUSPEND.

RB=PREVIOUS	Default: RB=PREVIOUS
RB=CURRENT	

Parameters

The parameters are explained as follows:

RB=PREVIOUS

RB=CURRENT

Specifies which RB on the TCB to suspend. The current RB is the one that is executing; it is the first RB on the RB chain. The previous RB is the one that follows the current RB on the RB chain.

ABEND Codes

070

See *z/OS MVS System Codes* for an explanation and programmer responses for this code.

Return and Reason Codes

None.

Example

Suspend the execution of the most recently chained request block of the current task.

SUSPEND RB=CURRENT

SUSPEND — Suspend Execution of an SRB

Description

Suspend an RB

To suspend an RB, use the variation of the SUSPEND macro described under “SUSPEND — Suspend Execution of an RB” on page 127.

To request suspension of a supervisor request block (SRB), use this variation of the SUSPEND macro.

When a caller issues the SUSPEND macro for an SRB, the system passes control to an exit routine identified on the SUSPEND macro and passes the suspend token to the routine. The exit routine decides whether to suspend the SRB or allow the SRB to continue execution and informs the system of its decision. If the SRB is to be suspended, the exit routine must store the suspend token so that the token can later be used to resume the SRB. The system takes the action requested by the exit routine. If the SRB is suspended, the SRB remains suspended until a subsequent RESUME macro either causes the SRB to resume execution or purges the SRB. If the exit routine allows the SRB to continue execution, control returns to the program that issued the SUSPEND macro.

Note: If the suspend completes successfully, the system will release any local lock that the caller might have held.

Environment

The requirements for the caller are:

Minimum authorization:	Supervisor state or PSW key 0 - 7
Dispatchable unit mode:	SRB
Cross memory mode:	Any
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for interrupts
Locks:	The caller may hold the local or CML lock but is not required to hold any
Control parameters:	Must be in the caller's primary address space or addressable through the caller's dispatchable unit access list (DU-AL)

Programming Requirements

Programming requirements for the calling program are:

- Before issuing the SUSPEND macro, ensure that the global symbol &SYSASCE is correctly set to indicate the ASC mode of your program. To test or set this global symbol, use the SYSSTATE macro.
- Programs in AR ASC mode must ensure that parameter addresses are ALET-qualified.

Restrictions

None.

SUSPEND Macro for SRBs

Input Register Information

Before issuing the SUSPEND macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter or using it as a base register.

Output Register Information

After the caller issues the macro, the system might use some registers as work registers or might change the contents of some registers. When the system returns control to the caller, the contents of these registers are not the same as they were before the macro was issued. Therefore, if the caller depends on these registers containing the same value before and after issuing the macro, the caller must save these registers before issuing the macro and restore them after the system returns control.

When control returns to the caller, the general purpose registers contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14	Used as a work register by the system
15	Return code

When control returns to the caller, the access registers contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

Performance Implications

None.

Syntax

The standard form of the SUSPEND macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
<i>b</i>	One or more blanks must precede SUSPEND.
SUSPEND	
<i>b</i>	One or more blanks must follow SUSPEND.

SPTOKEN= <i>sptoken addr</i>	<i>sptoken addr</i> : RX-type address.
,EXIT= <i>exit addr</i>	<i>exit addr</i> : RX-type address or register (2) - (12).
,EXITPARM= <i>exitparm addr</i>	<i>exitparm addr</i> : RX-type address.
,RSCODE= <i>rscode addr</i>	<i>rscode addr</i> : RX-type address.

`,RELATED=value`*value*: Any valid macro parameter specification.

Parameters

The parameters are explained as follows:

SPTOKEN=*sptoken addr*

Specifies the address of an 8-byte location where the system is to place the suspend token that identifies the SRB that is to be suspended.

,EXIT=*exit addr*

The 31-bit address of the suspend exit routine. The suspend exit routine must be addressable in the caller's primary address space.

,EXITPARM=*exitparm addr*

The 31-bit address of the parameters to be passed to the suspend exit routine.

,RSCODE=*rscod addr*

Specifies the address of the fullword where the system is to place the resume code optionally returned by the suspend exit routine or by the program that issued the RESUME macro.

,RELATED=*value*

Specifies information used to self-document macros by "relating" functions or services to corresponding functions or services. The format and content of the information provided is at the discretion of the user and may be any valid coding values.

ABEND Codes

017

See *z/OS MVS System Codes* for an explanation and programmer responses for this code.

Return Codes

When the SUSPEND macro returns control to your program, GPR 15 contains a hexadecimal return code.

Table 20. Return Codes for the SUSPEND Macro

Return Code	Meaning and Action
00	<p>Meaning: The SRB was successfully suspended and resumed. If you code the RSCODE parameter, the program that issues the RESUME macro might have stored a value into RSCODE.</p> <p>Action: None.</p>
04	<p>Meaning: The exit routine elected to allow the SRB to continue execution. If you coded the RSCODE parameter, the exit routine might have stored a value into RSCODE.</p> <p>Action: None.</p>

SUSPEND Macro for SRBs

Table 20. Return Codes for the SUSPEND Macro (continued)

Return Code	Meaning and Action
08	Meaning: Environmental error. A program tried to issue the SUSPEND macro from an SRB after the SRB abended with code X'47B'. This SRB cannot be suspended because it is in the process of being abended. Action: None required. However, you might take some action based upon your application.
0C	Meaning: Program error. A program tried to issue the SUSPEND macro from within the suspend exit. The suspend exit tried to resuspend the SRB. Action: Change your suspend exit program so it does not issue a SUSPEND request.
20	Meaning: Program error. An error occurred in the exit routine. Action: Correct your suspend exit program to remove program errors.
24	Meaning: A system error occurred. Action: Retry the request.

Example

Suspend the execution of an SRB.

```

:
:           SUSPEND  SPTOKEN=TOKEN,EXIT=EXITADD,RSCODE=RCODE
:
:
RCODE      DS      F'0'
EXITADD    DC      A(EXITRTN)
TOKEN      DS      CL8
:
:
```

SUSPEND (SRB)—List Form

For programs that require reentrant code, use the list form of the SUSPEND macro together with the execute form of the macro. The list form of the macro defines an area of storage that the execute form of the macro uses to store parameter values.

Syntax

The list form of the SUSPEND macro is valid only for suspending an SRB. It is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede SUSPEND.
SUSPEND	
b	One or more blanks must follow SUSPEND.

MF=L

,RELATED=*value* *value*: Any valid macro parameter specification.

Parameters

The parameters are explained under the standard form of the SUSPEND macro with the following exception:

MF=L
Requests the list form of SUSPEND.

SUSPEND (SRB)—Execute Form

For programs that require reentrant code, use the execute form of the SUSPEND macro together with the list form. The execute form of the macro stores the parameters into the storage area defined by the list form.

Syntax

The execute form of the SUSPEND macro is valid only for suspending an SRB. It is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede SUSPEND.
SUSPEND	
b	One or more blanks must follow SUSPEND.

SPTOKEN= <i>sptoken addr</i>	<i>sptoken addr</i> : RX-type address.
,EXIT= <i>exit addr</i>	<i>exit addr</i> : RX-type address or register (2) - (12).
,MF=(E, <i>cntl addr</i>)	<i>cntl addr</i> : RX-type address or register (2) - (12).
,EXITPARM= <i>exitparm addr</i>	<i>exitparm addr</i> : RX-type address.
,RSCODE= <i>rscod addr</i>	<i>rscod addr</i> : RX-type address.
,RELATED= <i>value</i>	<i>value</i> : Any valid macro parameter specification.

Parameters

The parameters are explained under the standard form of the SUSPEND macro with the following exception:

SUSPEND Macro for SRBs

,MF=(E,*cntl addr*)

Requests the execute form of SUSPEND. *cntl addr* must be the address of the parameter list provided by the list form of the macro.

SVCUPDTE — SVC Update

Description

Use the SVCUPDTE macro to dynamically replace or delete SVC table entries, or return the SVC number of a routine at a specified entry point. Callers who use this service are responsible for providing recovery. Improper deletion or replacement of system-provided SVC routines causes unpredictable results and might terminate the system.

The resource name, SYSZSVC TABLE, is available as the operand of an ENQ or DEQ macro, to be used when you must serialize the execution of a program that uses the SVCUPDTE macro.

See *z/OS MVS Programming: Authorized Assembler Services Guide* for additional information about the SVCUPDTE macro.

Environment

The requirements for the caller are:

Minimum authorization:	Supervisor state and PSW key 0
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	None

Programming Requirements

Ensure that the code for the SVC routine added to the SVC table has the correct attributes for the type of SVC specified.

The caller must include the CVT mapping macro.

Restrictions

None.

Input Register Information

Before issuing the SVCUPDTE macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter or using it as a base register.

Output Register Information

After the caller issues the macro, the system might use some registers as work registers or might change the contents of some registers. When the system returns control to the caller, the contents of these registers are not the same as they were before the macro was issued. Therefore, if the caller depends on these registers containing the same value before and after issuing the macro, the caller must save these registers before issuing the macro and restore them after the system returns control.

SVCUPDTE Macro

On input, register 13 must contain the address of a 72-byte save area.

When control returns to the caller, the general purpose registers (GPRs) contain:

Register	Contents
0	One of the following: <ul style="list-style-type: none">• If EXTRACT is specified: The SVC number• If REPLACE or DELETE is specified: Unchanged
1-13	Unchanged
14	Used as a work register by the system
15	Return code

Performance Implications

None.

Syntax

The SVCUPDTE macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
<i>b</i>	One or more blanks must precede SVCUPDTE.
SVCUPDTE	
<i>b</i>	One or more blanks must follow SVCUPDTE.

<i>num</i>	<i>num</i> : Symbol, decimal number, hexadecimal number (for example, X'02'), or register (2) - (12). Do not specify <i>num</i> with EXTRACT. Note: <i>num</i> cannot be 109, 116, 122, or 137 unless the ESR specification is also used.
,REPLACE ,DELETE ,EXTRACT ,EXTRACTANY	
,TYPE=1 ,TYPE=2 ,TYPE=3 ,TYPE=4 ,TYPE=6	Note: TYPE is only valid with REPLACE.
,EP= <i>addr</i>	<i>addr</i> : A-type address, decimal number, hexadecimal number, or register (2) - (12). <i>addr</i> should be a full 31-bit value with AMODE in bit 0.
,EPNAME= <i>entry-name</i>	<i>entry-name</i> : Symbol Note: EP and EPNAME are not needed with the DELETE option.
,LOCKS=(<i>lname</i> , <i>lname</i> ,...)	<i>lname</i> : CMS or LOCAL. Note: LOCKS is invalid with DELETE and EXTRACT, and cannot be specified with TYPE=6.

,APF=NO	Default: APF=NO
,APF=YES	Note: APF is only valid with REPLACE.
,AR=NO	Default: AR=NO
,AR=YES	Note: AR is valid only with REPLACE.
,NPRMPT=NO	Default: NPRMPT=NO
,NPRMPT=YES	Note: NPRMPT is only valid with REPLACE.
,RELATED= <i>value</i>	<i>value:</i> Any valid macro keyword specification.
,ESR= <i>esr</i>	<i>esr:</i> decimal number, or register (2) - (12).
,USEECVT=NO	Default: USEECVT=NO
,USEECVT=YES	

Parameters

The parameters are explained as follows:

num

Specifies the number of the SVC that is being inserted or deleted.

,REPLACE

,DELETE

Specifies the function to be performed. REPLACE indicates that an SVC table entry is to be inserted in the SVC table. This could be a new SVC or a replacement for an existing SVC. DELETE indicates that the specified SVC number is to be deleted from the SVC table.

If you issue an SVC instruction with a deleted or undefined SVC number, the program abnormally ends with a system completion code of X'Fnn' (*nn* is the operand of the SVC instruction, in hexadecimal). However, if you issue an SVCUPDTE macro using the DELETE parameter and specify a previously deleted SVC number, no abnormal end results.

,EXTRACT

Indicates that the user has supplied an EP or EPNAME and wishes to have the SVC number of that routine returned in register 0. The **num** parameter is not valid with this option.

,EXTRACTANY

Indicates that the user has supplied an EP or EPNAME and wishes to have the SVC or extended SVC number of that routine returned in register 0.

For a non-extended SVC

Bit 0 of register 0 has a value of 0. Bits 24–31 contain the SVC number.

For an extended SVC

Bit 0 of register 0 has a value of 1. Bits 16–23 contain the ESR number. Bits 24–31 contain the SVC number.

The **num** parameter is not valid with this option.

,TYPE=1

,TYPE=2

,TYPE=3

SVCUPDTE Macro

,TYPE=4

,TYPE=6

Specifies the SVC type for a REPLACE request. See the topic “Programming Conventions for SVC Routines” in *z/OS MVS Programming: Authorized Assembler Services Guide* for information concerning the characteristics and restrictions for each type of SVC.

,EP=addr

Specifies the entry point address of the SVC routine. The addressing mode of the entry point is defined by bit 0 of the entry point address of the SVC routine. If bit 0=1, the SVC routine will be entered in 31-bit addressing mode; if bit 0=0, the SVC routine will be entered in 24-bit addressing mode.

,EPNAME=entry-name

Specifies the entry name of the SVC routine. The entry name must be the load module name or alias of a module in LPA or the entry name of a module link edited into the nucleus. The AMODE of the SVC routine is determined when the SVC is link edited.

,LOCKS=(lname,lname,...)

Specifies the lock(s) required when the SVC routine executes. The lock(s) specified can be CMS or LOCAL. This parameter is valid only with REPLACE.

Notes:

1. TYPE=1 must not specify LOCAL.
2. TYPE=6 cannot specify any locks.
3. TYPE=2, 3, or 4 must specify LOCAL if CMS is specified.

,APF=YES

,APF=NO

Specifies whether or not the invocation of the SVC is to be restricted to authorized programs. This parameter is valid only with REPLACE.

,AR=YES

,AR=NO

Specifies whether or not the SVC can be issued by a program in access register mode. If you specify NO, a program that issues the SVC while in access register mode abends with a completion code of X'0F8'. This parameter is valid only with REPLACE.

,NPRMPT=YES

,NPRMPT=NO

Indicates whether or not the SVC can be preempted for I/O interruptions.

,RELATED=value

Provides information to document the macro by relating the function performed to another service or function. The format can be any valid coding value that the user chooses.

,ESR=esr

Specifies the extended SVC routing number of an extended SVC. You may supply a decimal number or a value in register (2) - (12). When you supply an explicit SVC number, the ESR parameter is only allowed with SVC numbers 109, 116, 122, and 137. When you provide the SVC number in a register, the ESR specification is ignored if the SVC number is not 109, 116, 122, or 137. This parameter is not valid with EXTRACT. When ESR is specified, the TYPE parameter is only used to validate other parameters, because each extended SVC has a predefined type that cannot be changed.

Note: When using SVC screening with the ESR parameter, the system ignores the screening information associated with the ESR number itself (for example, 109). The system only uses screening information associated with the routing code.

,USEECVT=YES

,USEECVT=NO

If you have verified that you are running OS/390 Release 10 or higher by checking the feature bits in the CVT, you may use this optional parameter to avoid some system processing. Specifying YES allows the system to locate the SVCUPDTE service with a pointer in the ECVT instead of using the NUCLKUP service. You must be running in AMODE 31 to use this parameter. This parameter also requires the IHAECVT mapping macro.

ABEND Codes

None.

Return Codes

When the SVCUPDTE macro returns control to your program, GPR 15 contains a hexadecimal return code.

Table 21. Return Codes for the SVCUPDTE Macro

Return Code	Meaning and Action
00	<p>Meaning: The macro completed successfully.</p> <p>Action: None.</p>
04	<p>Meaning: The macro was coded incorrectly. For example, the user requested REPLACE without specifying an SVC number.</p> <p>Action: Correct the error in the program that issued the macro. Verify that the execute form of the macro correctly references the list form.</p>
08	<p>Meaning: Program error. The DELETE parameter was not specified correctly.</p> <p>Action: Correct the error in the program that issued the request. Verify that the execute form of the macro correctly references the list form.</p>
0C	<p>Meaning: Program error. A REPLACE request contained incorrect information. For example, the user specified an SVC type that was not 1 through 6, or the specified entry point address was not on a halfword boundary.</p> <p>Action: Correct the error in the program that issued the request. Verify that the execute form of the macro correctly references the list form.</p>
10	<p>Meaning: Program error. A REPLACE request contained illogical information. For example:</p> <ul style="list-style-type: none"> • A type 6 SVC specified a lock. • Neither an entry point nor an EPNAME was provided for a REPLACE request. • Both an entry point and an EPNAME are provided. • The entry point provided is zero. • The CMS lock was requested without the LOCAL lock. <p>Action: Correct the error in the program that issued the request. Verify that the execute form of the macro correctly references the list form.</p>
14	<p>Meaning: Program error. The function specified was not REPLACE, DELETE, or EXTRACT.</p> <p>Action: Verify that the function specified is REPLACE, DELETE, or EXTRACT.</p>

SVCUPDTE Macro

Table 21. Return Codes for the SVCUPDTE Macro (continued)

Return Code	Meaning and Action
18	<p>Meaning: Program error. The user has attempted to update an extended SVC router entry in the SVC table (<i>num</i> was specified as 109, 116, 122, or 137).</p> <p>Action: Correct the error in the program that issued the macro. Verify that the execute form of the macro correctly references the list form.</p>
1C	<p>Meaning: Environmental error. Unable to locate the entry point address for an EPNAME specification.</p> <p>Action: Verify that all parts of the product or application are currently installed.</p>
20	<p>Meaning: Program error. An EXTRACT request contains illogical information. For example:</p> <ul style="list-style-type: none">• Neither an entry point address nor an EPNAME is specified.• Both an entry point address and an EPNAME are specified.• An SVC number is specified.• The entry point address specified is zero. <p>Action: Correct the error in the program that issued the macro. Verify that the execute form of the macro correctly references the list form.</p>
24	<p>Meaning: Environmental error. Unable to locate the SVC routine for the EXTRACT request.</p> <p>Action: Verify that all parts of the product or application are currently installed.</p>
28	<p>Meaning: System error. An error occurred while updating the SVC table.</p> <p>Action: Retry the request.</p>
44	<p>Meaning: Program error. A request was made to update an extended SVC, but no extended SVC routing code was provided.</p> <p>Action: When updating an extended SVC, use the ESR parameter to specify the extended SVC routing code.</p>
48	<p>Meaning: Program error. A request was made to update a non-extended SVC, but an extended SVC routing code was provided.</p> <p>Action: When updating a non-extended SVC, do not use the ESR parameter.</p>
52	<p>Meaning: Program error. A request was made to update an extended SVC, but the supplied SVC type did not match the system-defined type for that extended SVC.</p> <p>Action: When updating a non-extended SVC, use the TYPE parameter to specify the system-defined type for that particular extended SVC.</p>

Example 1

Delete SVC 200 from the SVC table.

```
SVCUPDTE 200,DELETE
```

Example 2

Insert SVC 201 in the SVC table. This is a type 2 SVC, with entry point at location SVCADDR. The SVC cannot be preempted for I/O interruptions.

```
SVCUPDTE 201,REPLACE,NPRMPT=NO,TYPE=2,EP=SVCADDR
```

Example 3

Replace SVC 202 in the SVC table. This is a type 1 SVC with entry point at the location in register 2.

SVCUPDTE 202,REPLACE,TYPE=1,EP=(2)

Example 4

Replace SVC 203 in the SVC table. SVC 203 is a type 4 SVC requiring the LOCAL lock. The routine has been loaded into LPA with the name MYSVC.

SVCUPDTE 203,REPLACE,TYPE=4,LOCKS=LOCAL,EPNAME=MYSVC

Example 5

Determine the SVC number associated with the name IGC062. The SVC number is to be returned in register 0.

SVCUPDTE ,EXTRACT,EPNAME=IGC062

Example 6

Replace SVC 202 in the SVC table. This is a type 3 SVC with entry point at explicit location X'FFEC00'. Note that this example uses a symbol as the SVC number.

SVCUPDTE SVCNUM,REPLACE,TYPE=3,EP=X'FFEC00'
.
.
.
SVCNUM EQU 202

SVCUPDTE—List Form

The list form of the SVCUPDTE macro builds a nonexecutable parameter list that can be referred to by the execute form of the SVCUPDTE macro.

Syntax

The list form of the SVCUPDTE macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede SVCUPDTE.
SVCUPDTE	
b	One or more blanks must follow SVCUPDTE.
<i>num</i>	<i>num</i> : Symbol, decimal number, hexadecimal number (for example X'02'). Note: This parameter must be specified on the execute and the list form of the macro. Do not specify <i>num</i> with EXTRACT.
,REPLACE	
,DELETE	
,EXTRACT	
,EXTRACTANY	
,TYPE=1	Note: TYPE is not valid with DELETE.
,TYPE=2	
,TYPE=3	

SVCUPDTE Macro

,TYPE=4

,TYPE=6

,EP=*addr*

addr: A-type address, decimal number, or hexadecimal number.

,EPNAME=*entry-name*

entry-name: Symbol

Note: EP and EPNAME are not needed with the DELETE option. This parameter must be supplied on either the execute or the list form.

,LOCKS=(*lname*, *lname*,...)

lname: CMS or LOCAL.

Note: This option is only valid with REPLACE and must not be specified with TYPE=6.

,APF=NO

Default: APF=NO

,APF=YES

Note: APF is only valid with REPLACE.

,AR=NO

Default: AR=NO

,AR=YES

Note: AR is valid only with REPLACE.

,NPRMPT=NO

Default: NPRMPT=NO

,NPRMPT=YES

Note: NPRMPT is only valid with REPLACE.

,RELATED=*value*

value: Any valid macro keyword specification.

,ESR=*esr*

esr: decimal number, or register (2) - (12).

,MF=L

Parameters

The parameters are explained under the standard form of the SVCUPDTE macro with the following exception:

,MF=L

Specifies the list form of the SVCUPDTE macro.

Example 1

Use the list form of the macro to replace SVC 202 in the SVC table. It is a type 2 SVC with entry point at location SVCADDR. The SVC routine needs the local lock.

SVCUPDTE 202,REPLACE,TYPE=2,LOCKS=LOCAL,MF=L,EP=SVCADDR

Example 2

Use the list form of the macro to replace SVC 201 in the SVC table. The routine is a type 2 SVC.

SVCUPDTE 201,REPLACE,TYPE=2,MF=L

SVCUPDTE—Execute Form

Syntax

The execute form of the SVCUPDTE macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
	One or more blanks must precede SVCUPDTE.
SVCUPDTE	
	One or more blanks must follow SVCUPDTE.

<i>num</i>	<i>num</i> : Register (2) - (12). Note: This parameter must be supplied on either the execute or the list form of the macro with REPLACE or DELETE, and it must not be specified with EXTRACT.
,EP= <i>addr</i>	<i>addr</i> : Register (2) - (12). Note: This parameter must be supplied on either the execute or the list form of the macro. This parameter is not needed with the DELETE option.
,EPNAME= <i>entry-name</i>	<i>entry-name</i> : Symbol Note: EP and EPNAME are not needed with the DELETE option. This parameter must be supplied on either the execute or the list form.
,RELATED= <i>value</i>	<i>value</i> : Any valid macro keyword specification.
,USEECVT=NO ,USEECVT=YES	Default: USEECVT=NO
,MF=(E, <i>addr</i>)	<i>addr</i> : RX-type address or register (2) - (12).

Parameters

The parameters are explained under the standard form of the SVCUPDTE macro with the following exception:

,MF=(E,*addr*)

Specifies the execute form of the SVCUPDTE macro.

Example

Use the execute form of the SVCUPDTE macro to perform the function specified by the parameter list whose address is given in register 2.

```
SVCUPDTE MF=(E,(2))
```

SVCUPDTE Macro

SWAREQ — Invoke SWA Manager in Locate Mode

Description

The SWAREQ macro has no standard form. It only has a list, an execute, and a modify form. The MF parameter, which indicates the form of the macro, is required.

When you invoke this macro in execute form, it uses the two parameters, FCODE and EPA, to modify the parameter list, which is at the location you specify by the *addr* value in the **MF=(E,addr)** parameter. After ensuring the validity of the parameters, it invokes the SWA manager in locate mode. The SWA manager obtains its input from the parameter list, and performs the function associated with the specified FCODE. If you do not specify any parameters, the macro assumes the parameter list already exists, and it simply invokes the SWA manager.

The modify form of SWAREQ is functionally the same as the execute form, except that the macro only modifies the parameter list without invoking the SWA manager. The list form of SWAREQ generates the parameter list that is modified by the other two forms of the macro, and it does not invoke the SWA manager.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state and any PSW key; see the UNAUTH=NO parameter description for an exception.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=SASN=HASN; see the UNAUTH parameter description for an exception.
AMODE:	24- or 31-bit; UNAUTH=YES requires 31-bit addressing mode.
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held; see the UNAUTH parameter description for an exception.
Control parameters:	Must be in the caller's primary address space

Programming Requirements

The caller must include the following mapping macros:

- IEFZB505
- IEFJESCT
- CVT
- IEFQMIDS

If you have specified through JES parameters that SWA is to be located above 16 megabytes, you must be in 31-bit addressing mode to access SWA. See *z/OS JES2 Initialization and Tuning Guide* or *z/OS JES3 Initialization and Tuning Guide* for information about locating SWA above 16 megabytes.

Restrictions

None.

SWAREQ Macro

Input Register Information

On input to the macro, register 13 must contain the address of a standard 18-word save area.

Output Register Information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register	Contents
0	When control returns from SWAREQ, used as a work register by the system. When control does not return from SWAREQ, the address of a 16-byte area containing: Bytes 1-4 Address of the QMPA Bytes 5-12 Not an intended programming interface; record this information and provide it to the appropriate IBM support personnel. Bytes 13-16 Address of the failing EPA
1	When control returns from SWAREQ, used as a work register by the system. When control does not return from SWAREQ, abend code 0B0.
2-12	Unchanged
13	If AMODE 31, then high bit will be cleared. If AMODE 24, then high byte will be cleared.
14	Unchanged
15	Return code

Performance Implications

None.

ABEND Codes

The caller of the SWAREQ macro might receive abend code X'0B0' with one of the following reason codes:

X'04'
X'08'
X'0C'
X'1C'
X'20'
X'24'
X'28'
X'34'

See *z/OS MVS System Codes* for explanations and responses for these codes.

Return and Reason Codes

The hexadecimal return code is in register 15. When you specify UNAUTH=YES, the return codes have the following meanings:

Table 22. Return Codes for SWAREQ, UNAUTH=YES

Return Code	Meaning
0	The SWAREQ service was successful.
8	The SVA in the SWA prefix was not valid.

Table 22. Return Codes for SWAREQ, UNAUTH=YES (continued)

Return Code	Meaning
24	The SVA does not correspond to any virtual address.
28	The pointer to the EPAL was not valid.

When you do not specify UNAUTH=YES, the return codes have the following meanings:

Table 23. Return Codes for the SWAREQ Macro

Return Code	Meaning
0	The SWAREQ service was successful.
8	The SVA in the SWA prefix was not valid.
0C	You attempted to read a block that was not yet written.
10	The length for a SWA block was not valid.
1C	The block id was not valid.
20	The block pointer was not valid.
24	The SVA does not correspond to any virtual address.

SWAREQ—List Form

Syntax

The list form of the SWAREQ macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede SWAREQ.
SWAREQ	
b	One or more blanks must follow SWAREQ.

,FCODE= <i>fnclde</i>	<i>fnclde</i> : Function code
,EPA= <i>addr</i>	<i>addr</i> : Address of the pointer to the EPA. In the list form, this address may only be specified symbolically.
,MF=L	

Parameters

The parameters are explained as follows:

,FCODE=fnclde

Specifies the function code for the locate mode request. Valid codes are:

SWAREQ Macro

RL Read/Locate
WL Write/Locate

For more information about the meaning of each code, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

,EPA=addr

Specifies the address of the pointer to the extended parameter area (EPA). Do not specify the EPA itself on the EPA parameter.

,MF=L

Specifies the list form of the SWAREQ macro.

SWAREQ—Execute Form

Syntax

The execute form of the SWAREQ macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede SWAREQ.
SWAREQ	
b	One or more blanks must follow SWAREQ.

,FCODE=fnclde	<i>fnclde</i> : Function code
,EPA=addr	<i>addr</i> : External parameter area pointer address. It may be specified symbolically, as a register enclosed in parentheses, or as a symbol equated to a register enclosed in parentheses.
,UNAUTH=YES ,UNAUTH=NO	Default: UNAUTH=NO.
,MF=(E,addr)	<i>addr</i> : RX-type address or register (1) - (12).

Parameters

The parameters are explained under the list form of the SWAREQ macro, with the following exceptions:

,UNAUTH=YES

,UNAUTH=NO

UNAUTH=YES specifies that the system is to invoke the unauthorized form of the SWA manager. The unauthorized form of the SWA manager provides the output of the RL function of the authorized SWA manager. If you also specify

the FCODE parameter, the SWAREQ macro checks the syntax of the FCODE parameter but does not use the function code.

To use SWAREQ with the default of UNAUTH=NO, you must be in supervisor state, holding no locks, in task mode, and not in cross memory mode. However, when you are using SWAREQ to perform a Read Locate, you can override these restrictions by specifying UNAUTH=YES. You must also issue the macro IEFZB505 LOCEPAX=YES, which generates a longer, 28 byte, EPA.

To use SWAREQ with UNAUTH=YES, you must be in 31-bit addressing mode.

,MF=(E,addr)

E specifies the execute form of the SWAREQ macro, and **addr** specifies the address of the parameter list.

SWAREQ—Modify Form

Syntax

The modify form of the SWAREQ macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede SWAREQ.
SWAREQ	
b	One or more blanks must follow SWAREQ.

,FCODE= <i>fnclde</i>	<i>fnclde</i> : Function code
,EPA= <i>addr</i>	<i>addr</i> : External parameter area pointer address. It may be specified symbolically, as a register enclosed in parentheses, or as a symbol equated to a register enclosed in parentheses.
,MF=(M, <i>addr</i>)	<i>addr</i> : RX-type address or register (1) - (12).

Parameters

The parameters are explained under the list form of the SWAREQ macro, with the following exceptions:

,MF=(M,addr)

M specifies the modify form of the SWAREQ macro, and *addr* specifies the address of the parameter list.

SWAREQ Macro

SWBTUREQ — Call SJF SWBTU Processing Services

Description

SWBTUREQ requests services for processing scheduler work block text units (SWBTUs). The RETRIEVE service can be requested on the SWBTUREQ macro. RETRIEVE obtains text unit information from SWBTUs for a specified set of keys and places the information in the output area defined by the caller.

An SWBTU is made up of JCL statement or dynamically created JCL information in contiguous text unit format. More than one SWBTU may be used to represent a single JCL statement.

Examples of the use of SWBTUREQ RETRIEVE are the JES sysout separator page installation exits. See the HASX15A member of SYS1.SAMPLIB for a sample exit.

For the RETRIEVE service, there are three calls that you can make:

- A call to determine the local working storage size needed for the service
- A call to determine the output area size needed to accommodate all the matched text units
- A call to obtain the text units that match the requested keys.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state, and any PSW key. For SWBTUREQ RETRIEVE, the caller must have a PSW key that matches the key of the caller's storage.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in the primary address space

Programming Requirements

The caller must set up recovery for SWBTUREQ RETRIEVE. The caller must include the CVT and IEFJESCT mapping macros. IEFJSJTRP maps the storage for the required RETRIEVE service parameter list. "SWBTUREQ RETRIEVE Input Parameters" on page 153 describes the parameter list's input fields. "SWBTUREQ RETRIEVE Output" on page 154 describes the fields that contain output on return from the SWBTUREQ RETRIEVE service.

The caller is responsible for supplying all storage for SWBTUREQ RETRIEVE processing. You can use SWBTUREQ RETRIEVE with different combinations of parameters to determine the local working storage size needed, and to determine and obtain the output area size needed for a particular request. A third combination of parameters allows you to invoke SWBTUREQ RETRIEVE to obtain text unit information. Table 24 on page 153 lists the required combination of parameters to use based on the type of service call you are making.

SWBTUREQ Macro

Restrictions

None.

Input Register Information

On input to the SWBTUREQ macro, the caller must insure that general purpose register (GPR) 13 points to a standard, 72-byte save area.

Output Register Information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register	Contents
0	If GPR 15 contains a zero, GPR 0 is used as a work register by the system. If GPR 15 contains return code 12, GPR 0 contains a reason code; otherwise, GPR 0 contains zero.
1	Used as a work register by the system
2-13	Unchanged
14	Used as a work register by the system
15	Return code

Performance Implications

None.

Syntax

The standard form of the SWBTUREQ macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede SWBTUREQ
SWBTUREQ	
b	One or more blanks must follow SWBTUREQ

REQUEST= <i>service</i>	<i>service</i> : Service name
,PARM= <i>addr</i>	<i>addr</i> : RX-type address, or registers (2) - (12). Register 1 is the default.

Parameters

The parameters are explained as follows:

REQUEST=*service*

Specifies the SJF SWBTU service to be called. RETRIEVE is the valid service name.

,PARM=*addr*

Specifies the address of the parameter list for the service requested. The

parameter list for the RETRIEVE service is IEFSJTRP. “SWBTUREQ RETRIEVE Input Parameters” lists the parameter fields you must initialize.

SWBTUREQ RETRIEVE Service

Use RETRIEVE to obtain text unit information for a specified set of JCL or output descriptor keys. The retrieved information is placed in a caller-defined output area.

Table 24. Parameter Combinations for SWBTUREQ RETRIEVE Functions

Function	Required Parameters
Obtain local working storage size	SJTRID, SJTRVERS, SJTRLEN, SJTRSTOR, SJTRSTSZ, SJTRAREA, and SJTRSIZE. SJTRSTOR, SJTRSTSZ, SJTRAREA, and SJTRSIZE should be zero on this invocation.
Obtain output area size	SJTRID, SJTRVERS, SJTRLEN, SJTRSTOR, SJTRSTSZ, SJTRSWBN, SJTRSWBA, SJTRKIDN, SJTRKIDL. Fill in SJTRSTOR and SJTRSTSZ with the values returned when you invoked the macro to determine the local working storage size. SJTRAREA and SJTRSIZE should be zero on this invocation.
Retrieve requested keys	SJTRID, SJTRVERS, SJTRLEN, SJTRSTOR, SJTRSTSZ, SJTRSWBN, SJTRSWBA, SJTRKIDN, SJTRKIDL. Fill in SJTRSTOR, SJTRSTSZ, SJTRAREA, and SJTRSIZE with the values returned when you invoked the macro to determine the local working storage size and the output buffer size.

SWBTUREQ RETRIEVE Input Parameters

For each SWBTUREQ invocation, you need to initialize certain fields of parameter list IEFSJTRP. Figure 4 on page 155 illustrates some of the parameter fields and their relationships to other fields. The list below describes the valid value assignments for all input parameters in IEFSJTRP.

SJTRID

The identifier ‘SJTR’ of the SWBTUREQ RETRIEVE parameter list. Assign the symbolic equate SJTRCID to this field.

SJTRVERS

The current version number of the SWBTUREQ RETRIEVE service. Assign the symbolic equate SJTRCVER to this field.

SJTRLEN

The length of the SWBTUREQ RETRIEVE parameter list (IEFSJTRP). Assign the symbolic equate SJTRLGTH to this field.

SJTRSTOR

The local working storage pointer or zero.

SJTRSTSZ

The size of the local working storage area required by the service.

SJTRSWBN

The number of SWBTUs in the SWBTU address list table. The table is mapped by SJTRSBTL.

SJTRSWBA

The address of the SWBTU address list table from which text units are retrieved. The address list is mapped by SJTRSBTL.

SJTRAREA

The address of the text unit output area.

SWBTUREQ Macro

SJTRSIZE

The size of the text unit output area.

SJTRKIDN

The number of entries in the key list. The key list is mapped by SJTRKEYL.

SJTRKIDL

The address of the list of keys that are to be retrieved. The key list is mapped by SJTRKEYL.

SJTRSBTL

SWBTU address list from which text units are returned. The list contains one entry per SWBTU. Parameter SJTRSWBN specifies the number of entries in the list; SJTRSWBA specifies the address of the list. More than one SWBTU may be used to represent a single JCL statement.

SJTRSTUP

An SWBTU address entry.

SJTRKEYL

The requested list of keys to be retrieved. Parameter SJTRKIDN specifies the number of entries in the list; SJTRKIDL specifies the address of this list.

SJTRKYID

The key to be used for the retrieve. If you are using the RETRIEVE service to obtain information about output descriptors, the key values for the attributes are defined in mapping macro IEFDOKEY. See *z/OS MVS Data Areas, Vol 2 (DCCB-ITZYRETC)* for the mapping provided by IEFDOKEY. *z/OS MVS Programming: Authorized Assembler Services Guide* lists the dynamic output keys and their JCL equivalents.

SWBTUREQ RETRIEVE Output

These parameters are returned with values on completion of RETRIEVE service processing.

SJTRREAS

The reason code returned. The reason codes are defined in "Return and Reason Codes" on page 155.

SJTRWKSZ

The local working storage size required by the SWBTUREQ service.

SJTRTULN

The size of the area needed to contain all matched requested text units.

SJTRERRP

This field contains a zero unless a parameter list error (key list or SWBTU address entry error) occurs. In the case of a key list error (return code 8 with a reason code of 65), the address of the key list entry in error appears in the field. In the case of a SWBTU address entry error (return code 8 with a reason code of 19 or 28), the address of the SWBTU address entry in error appears in the field.

SJTRAREA

The address of the text unit output area. The text unit output area contains the matched text unit strings organized by key in the order they were requested. Matched text unit strings are contiguous in the text unit output area. For the mapping of each text unit see the IEFDOTUM mapping macro in *z/OS MVS Data Areas, Vol 2 (DCCB-ITZYRETC)*.

SWBTUREQ Macro

Table 25. Return and Reason Codes for SWBTUREQ RETRIEVE

Return Code	Reason Code	Meaning and Action
00	None	<p>Meaning: RETRIEVE processing completed successfully. At least one text unit and key match was found. For any unmatched text units, the corresponding SJTRTPAD values are zero.</p> <p>Action: None.</p>
04	None	<p>Meaning: Either the caller did not provide enough storage for the service or none of the requested items was found. This is the expected return code if you issue the RETRIEVE request to obtain the working storage size or output area size.</p>
	004	<p>Meaning: RETRIEVE requires more local working storage. The size of the local working storage, as specified in input parameter SJTRSTSZ, is not large enough for the service. The amount of local working storage needed appears in output parameter SJTRWKSZ.</p> <p>Action: Repeat the request and supply the amount of storage returned in SJTRWKSZ in SJTRSTSZ.</p>
	008	<p>Meaning: RETRIEVE requires more text unit output area storage. The size needed for the output storage area appears in output parameter SJTRTULN.</p> <p>Action: Repeat the request and supply the amount of storage returned in SJTRTULN in SJTRAREA.</p>
	064	<p>Meaning: Successful completion. None of the keys in the input key list were found in the SWBTUs. All corresponding SJTRTPAD values are zero. There are no returned text units in the output area.</p> <p>Action: None required.</p>
08	None	<p>Meaning: The parameter list is not valid.</p>
	015	<p>Meaning: The parameter length specified in SJTRLEN is not valid for the specified version.</p>
	016	<p>Meaning: The version number specified is not correct for this service.</p>
	018	<p>Meaning: The caller must provide at least one SWBTU. Input parameter SJTRSWBN must be greater than zero, and SJTRSWBA must reference the SWBTU address list.</p> <p>Action: Set SJTRSWBN to a value greater than zero; set SJTRSWBA to the address of the SWBTU address list.</p>
	019	<p>Meaning: The specified SWBTU is not valid. Either one of the entries is zero, or the SWBTU address entry is not valid. Output parameter SJTRERRP contains the address of the SWBTU address entry.</p> <p>Action: Specify a valid SWBTU referenced by the SWBTU address is in SJTRERRP.</p>

Table 25. Return and Reason Codes for SWBTUREQ RETRIEVE (continued)

Return Code	Reason Code	Meaning and Action
	01A	<p>Meaning: Program error. The text length shown in the DOCNTLEN field in the IEFDOTUM mapping macro is not valid. SWBTUREQ RETRIEVE processing stops. One possible cause of the problem is a storage overlay.</p> <p>Action: If your program has overlaid storage, correct the error and rerun the program. Otherwise, contact the appropriate IBM support personnel.</p>
	028	<p>Meaning: Either all of the verbs for the SWBTUs or all of the labels for the SWBTUs do not match. Output parameter SJTRERRP contains the address of the SWBTU address entry where the inconsistency was found.</p> <p>Action: Correct the SWBTU in the SWBTU address table entry whose address is in SJTRERRP.</p>
	029	<p>Meaning: The output area size, defined by the combination of the input parameters SJTRAREA and SJTRSIZE, is not valid. One of the parameters is zero and the other is not zero.</p> <p>Action: Set SJTRAREA and SJTRSIZE to values greater than 0. (See "SWBTUREQ RETRIEVE Input Parameters" on page 153.)</p>
	065	<p>Meaning: The key entry is not valid. Input parameter SJTRKYID is zero. Output parameter SJTRERRP contains the address of the error key entry, SJTRKYID.</p> <p>Action: Correct the key in error (the key pointed to by SJTRKYID).</p>
	066	<p>Meaning: At least one key must be requested. Input parameter SJTRKIDN must be greater than zero and SJTRKIDL must reference the key entry list.</p> <p>Action: Set SJTRKIDN and SJTRKIDL to values greater than zero. (See input parameter description.)</p>
0C	None	<p>Meaning: A severe parameter list error occurred. The reason code appears in register 0.</p>
	014	<p>Meaning: An incorrect parameter ID was specified. SJTRID is not 'SJTR'.</p> <p>Action: See "SWBTUREQ RETRIEVE Input Parameters" on page 153 for SJTRID.</p>
	015	<p>Meaning: An incorrect parameter length was specified. SJTRLEN is not at least as large as the common parameter list size, 36 bytes.</p> <p>Action: See "SWBTUREQ RETRIEVE Input Parameters" on page 153 for SJTRLEN.</p>
	016	<p>Meaning: The version number is not correct. SJTRVERS is less than one.</p> <p>Action: See "SWBTUREQ RETRIEVE Input Parameters" on page 153 for SJTRVERS.</p>

SWBTUREQ Macro

Table 25. Return and Reason Codes for SWBTUREQ RETRIEVE (continued)

Return Code	Reason Code	Meaning and Action
	017	Meaning: The service specified by SWBTUREQ REQUEST=service, is not known. Action: Specify a valid request for 'service'.
14	None	Meaning: SJF encountered a condition that would have caused an abend. If processing had continued, an abend would have occurred.
18	None	Meaning: The service routines for SWBTUREQ are not available.

Example

Invoke the SWBTUREQ RETRIEVE service to obtain text unit information for three output descriptor attributes (title, name, and room). Represent the output descriptor with SWBTU1 and SWBTU2. Use register 6 to contain the address of SWBTU1, and register 7 to contain the address of SWBTU2. Use AREA to define the text unit output area. Define the service's local working storage in LCLSTOR. Establish an ESTAE for a recovery environment.

On return, register 15 contains return code 0, register 0 contains reason code 0, and SJTRREAS contains reason code 0. The output areas from the service contain the following information:

- SJTRAREA contains two contiguous text units that were in the SWBTUs and were requested in the key list.
- SJTRWKSZ contains the size of local working storage required for the SWBTUREQ RETRIEVE service.
- SJTRTULN contains the size of the area used to return the two matched text units.
- SJTRKEYL contains unchanged SJTRKYIDs.

SJTRTPAD contains the pointers for the matched text units and zero for unmatched text units. For this example, SJTRTPAD's first entry points to the first text unit returned in the text unit output area. The second entry contains zero, and the third entry points to the second text unit returned in the text unit output area.

```
*****
*
*      Fill in 3 requested keys in the key list, SJTRKEYL.
*
*****
*
*      XC      KEYLIST,KEYLIST      Initialize KEYLIST area
*      LA      R2,KEYLIST           Point to start of key list
*      USING   SJTRKEYL,R2         Establish addressability
*
*      MVC     SJTRKYID+0*SJTRKLEN,=Y(DOTITLE)  Title Key
*      MVC     SJTRKYID+1*SJTRKLEN,=Y(DONAME)   Name Key
*      MVC     SJTRKYID+2*SJTRKLEN,=Y(DOROOM)   Room Key
*
*****
*
*      Fill in 2 SWBTU pointers in SWBTU address list, SJTRSBTL.
*
*****
*
*      XC      SWBTULST,SWBTULST      Initialize SWBTU pointer list area
*
```

SWBTUREQ Macro

```

        LA    R3,SWBTULST      Point to start of SWBTU pointer list
        USING SJTRSBTL,R3      Establish addressability
*
        ST    R6,SJTRSTUP      Set address of first SWBTU in 1st
*                               entry of SWBTU address list
        LA    R3,SJTRSLN(,R3)  Point to second entry in SWBTU
        ST    R7,SJTRSTUP      Set address of second SWBTU in 2nd
*                               entry of SWBTU address list
*
*****
*                               *
*       Fill in the SWBTUREQ RETRIEVE parameter list, IEFSJTRP.      *
*                               *
*****
*
        XC    SJTRP(SJTRLGTH),SJTRP  Clear the parameter list
        MVC   SJTRID,=A(SJTRCID)    Parameter list ID
        MVI   SJTRVERS,SJTRCVER     Parameter list version
        LA    R4,SJTRLGTH           Get parameter list length
        STH   R4,SJTRLEN            Set parameter list length
*
        LA    R4,LCLSTOR           Get local working storage address
        ST    R4,SJTRSTOR          Set local working storage address
        LA    R4,STORLGTH          Get local working storage size
        STH   R4,SJTRSTSZ          Set local working storage size
*
        LA    R4,AREA              Get text unit output area address
        ST    R4,SJTRAREA          Set text unit output area address
        LA    R4,AREALGTH          Get text unit output area size
        STH   R4,SJTRSIZE          Set text unit output area size
*
        ST    R2,SJTRKIDL          Set key list address
        LA    R4,KEYNUM            Get number of requested keys
        STH   R4,SJTRKIDN          Set number of request keys
*
        LA    R3,SWBTULST          Point to start of SWBTU pointer list
        ST    R3,SJTRSWBA          Set address of SWBTU address list
        LA    R4,SWBTUNUM          Get number of SWBTUs
        STH   R4,SJTRSWBN          Set number of SWBTUs
*
*****
*                               *
*       Set up Register 1 to point to a word of storage that        *
*       contains the address of the parameter list, IEFSJTRP.      *
*                               *
*****
*
        LA    R4,SJTRP             Address of
        ST    R4,SJTRPPTR          the SWBTUREQ RETRIEVE
        LA    R1,SJTRPPTR          parameter list
*
*****
*                               *
*       Invoke the SWBTUREQ macro to retrieve the matched text units *
*       for items in the requested key list.                        *
*                               *
*****
*
        SWBTUREQ REQUEST=RETRIEVE  Issue the SJF macro
*
*****
*                               *
*       Check for a zero return code.                                *
*                               *
*****
*
        LTR   R15,R15             Check service return code

```

SWBTUREQ Macro

```

        BNZ    NODATA          Go to nonzero return processing
*
*      Code to process zero return code from SWBTUREQ ...
*      .
*      .
*      .
*
NODATA  DS      0H              Label used for branch when SWBTUREQ
*                                service returns with a nonzero
*                                return code.
*
*      Code to process nonzero return code from SWBTUREQ...
*      .
*      .
*      .
*
*****
*
*      Storage definitions
*
*****
*
        IEFSJTRP DSECT=NO      SWBTUREQ Retrieve parameter list
*
        SJTRPPTR DS      A      Field used to contain SJTRP address
*
        KEYLIST  DS      CL24    Area mapped by SJTRKEYL:
*                                enough storage for 3 entries
*
        SWBTULST DS      CL16    Area mapped by SJTRSBTL:
*                                enough storage for 2 entries
*
        AREA     DS      CL600   Area used by SWBTUREQ service to move
*                                matched text units for output.
*                                600 bytes is large enough and was
*                                chosen at random.
        AREALGTH EQU    *-AREA   Size of AREA
*
        LCLSTOR  DS      CL1000  Area used by SWBTUREQ service as
*                                local working storage.
        STORLGTH EQU    *-LCLSTOR Size of LCLSTOR
*
*****
*
*      Equates and Constants
*
*****
*
        R0       EQU    0        Register 0
        R1       EQU    1        Register 1
        R2       EQU    2        Register 2
        R3       EQU    3        Register 3
        R4       EQU    4        Register 4
        R15      EQU    15       Register 15
*
        SWBTUNUM EQU    2        Indicates number of SWBTUs
        KEYNUM   EQU    3        Indicates number of requested keys
*
        IEFDOKEY          Dynamic Output keys
        CVT   DSECT=YES   CVT mapping macro
        IEFJESCT          IEFJESCT mapping macro

```

SYNCH and SYNCHX — Take a Synchronous Exit to a Processing Program

Description

The SYNCH macro takes a synchronous exit to a processing program. After the processing program has been executed, the program that issued the SYNCH macro regains control. The SYNCH macro is also described in *z/OS MVS Programming: Assembler Services Reference ABE-HSP* with the exception of the KEYADDR, STATE, KEYMASK, and XMENV parameters. These parameters are restricted to programs in supervisor state, key 0-7, or APF-authorized.

If you are executing in 31-bit addressing mode, you must use the MVS/SP Version 2 of this macro, or a later version.

The SYNCH macro is intended for use by primary mode programs only. If your program runs in access register (AR) mode, use SYNCHX, which provides the same function as SYNCH. Descriptions of SYNCH and SYNCHX in this book are:

- The standard form of the SYNCH macro, which includes general information about the SYNCH and SYNCHX macros and some specific information about the SYNCH macro. The syntax of the SYNCH macro is presented, and all SYNCH parameters are explained.
- The standard form of the SYNCHX macro, which presents information specific to the SYNCHX macro and callers in AR mode.
- The list form of the SYNCH and SYNCHX macros.
- The execute form of the SYNCH and SYNCHX macros.

If the caller is in AR mode, the system passes the following values, unchanged, to the processing program:

- ARs 0-13
- Bits 16 and 17 of the current PSW indicating the ASC mode (primary or AR mode, where primary=secondary=home)
- Extended authorization index (EAX)

Parameters for SYNCH and SYNCHX must be in the caller's primary address space. Callers in AR mode must initialize AR 1 to zero before issuing SYNCHX.

Register Information

After the caller issues the macro, the system might use some registers as work registers or might change the contents of some registers. When the system returns control to the caller, the contents of these registers are not the same as they were before the macro was issued. Therefore, if the caller depends on these registers containing the same value before and after issuing the macro, the caller must save these registers before issuing the macro and restore them after the system returns control.

When control returns to the caller, the GPRs contain:

Register	Contents
0-1	Values the processing program placed there before it returned to the caller
2-13	If RESTORE=YES, unchanged

SYNCH and SYNCHX Macros

	If RESTORE=NO, values the processing program placed there before it returned to the caller
14	Used as a work register by the system
15	Value the processing program placed there before it returned to the caller

Syntax

The SYNCH macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
<i>b</i>	One or more blanks must precede SYNCH.
SYNCH	
<i>b</i>	One or more blanks must follow SYNCH.

<i>entry point addr</i>	<i>entry point addr</i> : RX-type address, or register (2) - (12) or (15).
,RESTORE=NO ,RESTORE=YES	Default: RESTORE=NO
,KEYADDR= <i>addr</i> ,KEYADDR=NOKEYADDR	<i>addr</i> : RS-type address, or register (2) - (12). Default: KEYADDR=NOKEYADDR (The key in the TCB is used.)
,STATE=PROB ,STATE=SUPV	Default: STATE=PROB
,KEYMASK= <i>addr</i> ,XMENV= <i>addr</i>	<i>addr</i> : RX-type address, or register (0) - (12). <i>addr</i> : RX-type address or register (0) - (12).
,AMODE=24 ,AMODE=31	Default: AMODE=CALLER
,AMODE=DEFINED ,AMODE=CALLER	Note: AMODE=DEFINED can only be specified if the entry point is provided in a register.

Parameters

The parameters are explained as follows:

entry point addr
Specifies the address of the entry point of the processing program to receive control.

,RESTORE=NO

,RESTORE=YES

Specifies whether registers 2-13 are to be restored when control is returned to the issuer of SYNCH.

,KEYADDR=addr

,KEYADDR=NOKEYADDR

addr specifies the address of a one-byte area that contains the key in which the exit is to receive control. The key must be in bits 0-3; bits 4-7 must be zero. If KEYADDR=*addr* is not specified, the key in the TCB is used as the default.

,STATE=PROB

,STATE=SUPV

Specifies the state in which the requested program receives control. PROB specifies problem state and SUPV specifies supervisor state.

,KEYMASK=addr

Specifies the address of a halfword, which along with the protect key of the currently active TCB, will be an operand in an OR instruction. The results of that instruction produce the PKM of the routine to which your program will take a synchronous exit.

If you specify KEYMASK, do not specify XMENV.

,XMENV=addr

Specifies the address of a parameter list that the caller passes to the SYNCH macro service. The parameter list contains values that set up a cross memory environment for the new PRB. The parameter list consists of a 10-byte list of values that determine some of the characteristics the PRB will have when it receives control. The parameter list must reside in the primary address space and the AR that qualifies the address must be set to zero. The format of the parameter list is as follows:

Bytes	Content of field
0-1	The value X'0A'
2-3	PKM value, which along with the protect key of the currently active TCB, will be an operand in an OR instruction. The results of that instruction produce the PKM of the routine to which the synchronous is to be taken.
4-5	SASN - defining the secondary address space for the exit routine
6-7	Extended authorization index (EAX) for the exit routine
8-9	PASN - defining the primary address space for the exit routine

If you specify XMENV, do not specify KEYMASK.

,AMODE=24

,AMODE=31

,AMODE=DEFINED

,AMODE=CALLER

Specifies the addressing mode in which the requested program is to receive control.

If AMODE=24 is specified, the requested program will receive control in 24-bit addressing mode.

If AMODE=31 is specified, the requested program will receive control in 31-bit addressing mode.

If AMODE=DEFINED is specified, the user must provide the entry point using a register, not an RX-type address. The requested program will receive control in

SYNCH and SYNCHX Macros

the addressing mode indicated by the high-order bit of the entry point address. If the bit is off, the requested program will receive control in 24-bit addressing mode; if the bit is set, the requested program will receive control in 31-bit addressing mode.

If AMODE=CALLER is specified, the requested program will receive control in the addressing mode of the caller.

Example 1

Take a synchronous exit to a processing program whose entry point address is specified in register 8.

```
SYNCH (8)
```

Example 2

Take a synchronous exit to a processing program labeled SUBRTN and restore registers 2-13 when control is returned.

```
SYNCH SUBRTN,RESTORE=YES
```

Example 3

Take a synchronous exit to a processing program whose entry point address is specified in register 5, modify the program's protect key by the KEYADDR and KEYMASK values, and restore registers 2-13 when control returns.

```
SYNCH (5),RESTORE=YES,KEYADDR=KEYBYTE,KEYMASK=MSKADDR
      .
      .
      .
KEYBYTE DC X'80'
MSKADDR DC X'0080'
```

Example 4

Take a synchronous exit to the program located at the address given in register 8 and restore registers 2-13 when control returns. Indicate that this program is to execute in 24-bit addressing mode.

```
SYNCH (8),RESTORE=YES,AMODE=24
```

SYNCHX — Take a Synchronous Exit to a Processing Program

The SYNCHX macro allows a program running in primary or AR mode to take a synchronous exit to a processing program. This macro is the same as the SYNCH macro, except that, for callers in AR mode, it generates code and addresses that are appropriate in AR mode. All parameters on the SYNCH macro are valid for the SYNCHX macro.

You can issue the SYNCHX macro in 64-bit addressing mode. However, AMODE=DEFINED can only be used to SYNCHX to amode 24 or amode 31 programs.

Before you issue the SYNCHX macro, issue the SYSSTATE ASCENV=AR macro to tell the SYNCHX macro to generate code appropriate for AR mode.

Syntax

The SYNCHX macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede SYNCHX.
SYNCHX	
b	One or more blanks must follow SYNCHX.

<i>entry point addr</i>	<i>entry point addr</i> : RX-type address, or register (2) - (12) or (15).
,RESTORE=NO ,RESTORE=YES	Default : RESTORE=NO
,KEYADDR= <i>addr</i> ,KEYADDR=NOKEYADDR	<i>addr</i> : RX-type address, or register (2) - (12). Default : KEYADDR=NOKEYADDR (The key in the TCB is used.)
,STATE=PROB ,STATE=SUPV	Default : STATE=PROB
,KEYMASK= <i>addr</i> ,XMENV= <i>addr</i>	<i>addr</i> : RX-type address, or register (0) - (12). addr : RX-type address or register (0) - (12).
,AMODE=24 ,AMODE=31 ,AMODE=64 ,AMODE=DEFINED	Default : AMODE=CALLER Note : AMODE=DEFINED can only be specified if the entry point is provided in a register. AMODE=DEFINED can only be used to SYNCHX to amode 24 and amode 31 programs.
,AMODE=CALLER	

Parameters

The parameters are described under the syntax of the standard form of the SYNCH macro.

SYNCH and SYNCHX—List Form

The list form of the SYNCH or SYNCHX macro is used to construct a control program parameter list.

Syntax

The list form of the SYNCH or SYNCHX macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede SYNCH or SYNCHX.

SYNCH and SYNCHX Macros

SYNCH or SYNCHX

b One or more blanks must follow SYNCH or SYNCHX.

,RESTORE=NO
,RESTORE=YES

Default: RESTORE=NO

,STATE=PROB
,STATE=SUPV

Default: STATE=PROB

,KEYMASK=*addr*
,XMENV=*addr*

addr: A-type address.
addr: RX-type address or register (0) - (12).

,AMODE=24
,AMODE=31
,AMODE=DEFINED
,AMODE=CALLER

Default: AMODE=CALLER

,MF=L

Parameters

The parameters are explained under the standard form of the SYNCH macro with the following exception:

,MF=L

Specifies the list form of the SYNCH macros.

Example

Use the list form of the SYNCH macro to specify that registers 2-13 are to be restored when control returns from executing the SYNCH macro and that the addressing mode of the program is to be defined by the high-order bit of the entry point address. Assume that the execute form of the macro specifies the program address.

SYNCH ,RESTORE=YES,AMODE=DEFINED,MF=L

SYNCH and SYNCHX—Execute Form

The execute form of the SYNCH or SYNCHX macro uses a remote program parameter list that can be generated by the list form of SYNCH or SYNCHX.

Syntax

The execute form of the macro is written as follows:

name

name: Symbol. Begin *name* in column 1.

↳ One or more blanks must precede SYNCH or SYNCHX.

SYNCH

↳ One or more blanks must follow SYNCH or SYNCHX.

<i>entry point addr</i>	<i>entry point addr</i> : RX-type address, or register (2) - (12) or (15).
,RESTORE=NO ,RESTORE=YES	
,KEYADDR= <i>addr</i> ,KEYADDR=NOKEYADDR	<i>addr</i> : RX-type address, or register (2) - (12).
,STATE=PROB ,STATE=SUPV	
,KEYMASK= <i>addr</i> ,XMENV= <i>addr</i>	<i>addr</i> : RX-type address, or register (0) - (12). <i>addr</i> : RX-type address or register (0) - (12).
,AMODE=24 ,AMODE=31 ,AMODE=DEFINED ,AMODE=CALLER	Note: AMODE=DEFINED can only be specified if the entry point is provided in a register.
,MF=(E, <i>ctrl addr</i>)	<i>ctrl addr</i> : RX-type address or register (1), (2) - (12).

Parameters

The parameters are explained under the standard form of the SYNCH macro with the following exceptions:

,KEYADDR=NOKEYADDR

Indicates that the default(the key in the TCB) should be used instead of the key in the parameter list defined by a list form of the macro.

,MF=(E,*ctrl addr*)

Specifies the execute form of the SYNCH macro using a list generated by the list form of SYNCH.

Example

Use the execute form of the SYNCH macro to take a synchronous exit to the program located at the address given in register 8 and restore registers 2-13 when control returns. Indicate that the program is to receive control in the same addressing mode as the caller and that the parameter list is located at SYNCHL2.

```
SYNCH (8),RESTORE=YES,AMODE=CALLER,MF=(E,SYNCHL2)
```

SYNCH and SYNCHX Macros

SYSEVENT — System Event

Description

The SYSEVENT macro provides the interface to the system resource manager (SRM). Using SYSEVENT mnemonics, you can notify SRM of an event or ask SRM to perform a specific function.

Environment

The requirements for the ENTRY=BRANCH caller are:

Authorization:	Supervisor state or PSW key 0 - 7
Dispatchable unit mode:	Task or SRB.
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	Callers that specify ENTRY=BRANCH must hold the LOCAL lock for TRAXRPT, TRAXFRPT and TRAXERPT. There are no locking requirements for the other SYSEVENTs.

The requirements for the ENTRY=SVC caller are:

Authorization:	APF authorized
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locking requirements

The following SYSEVENTs are unauthorized:

FREEAUX
QVS
REQFASD

The requirements are:

Authorization:	Problem state, with any PSW key
AMODE:	31-bit

All other requirements are as noted above for ENTRY=BRANCH and ENTRY=SVC.

Programming Requirements

When you specify ENTRY=BRANCH, include the CVT mapping macro as a DSECT in the calling program. If a specific SYSEVENT requires a parameter list in addition to the information specified on the macro invocation, load register 1 with the address of that parameter list before issuing the macro.

Restrictions and Limitations

Restrictions on the use of each SYSEVENT, including input and output requirements, follow the descriptions of the parameters.

SYSEVENT Macro

Input Register Information

When you specify ENTRY=BRANCH, register 13 must contain the address of a 72-byte save area on input. For specific input register requirements, see the description of the specific SYSEVENT.

Output Register Information

After the caller issues the macro, the system might use some registers as work registers or might change the contents of some registers. When the system returns control to the caller, the contents of these registers are not the same as they were before the macro was issued. Therefore, if the caller depends on these registers containing the same value before and after issuing the macro, the caller must save these registers before issuing the macro and restore them after the system returns control.

When control returns to the caller, the general purpose registers (GPRs) contain:

Register	Contents
0	Used as a work register by the system
1	One of the following: <ul style="list-style-type: none">• Unchanged, for ENCASSOC, STGTEST, TRAXRPT, TRAXFRPT, TRAXERPT, REQASCL, REQASD, REQFASD, REQSRMST, ENQHOLD, ENQRLSE, and QVS SYSEVENTs• Status code for DONTSWAP, OKSWAP, and TRANSWAP SYSEVENTs
2-13	Unchanged
14	Used as a work register by the system
15	One of the following: <ul style="list-style-type: none">• Return code, for ENCASSOC, TRAXRPT, TRAXFRPT, TRAXERPT, REQASCL, REQASD, REQFASD, REQSRMST, ENQHOLD, ENQRLSE, and QVS SYSEVENTs• Used as a work register by the system, for DONTSWAP, OKSWAP, STGTEST, and TRANSWAP SYSEVENTs

Syntax

The SYSEVENT macro is written as follows:

<i>name</i>	<i>name</i> : symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede SYSEVENT.
SYSEVENT	
b	One or more blanks must follow SYSEVENT.

<i>sysevent mnemonic</i>	<i>sysevent mnemonic</i> : symbol. Note: See the description of the parameters for the valid options.
--------------------------	---

`,ENTRY=SVC`**Defaults:**

ENTRY=BRANCH for the following SYSEVENTs:

- ENCASSOC
- ENCSTATE
- QVS
- TRAXERPT
- TRAXFRPT
- TRAXRPT
- REQFASD

`,ENTRY=BRANCH`

ENTRY=SVC for the following SYSEVENTs:

DONTSWAP	OKSWAP
TRANSWAP	STGTEST
REQASCL	REQASD
REQSRMST	ENQHOLD
ENQRLSE	REQLPDAT

`,TYPE=BLOCK``,TYPE=BYTE``,TYPE=2`

Note: TYPE=BLOCK and TYPE=BYTE are valid only for SYSEVENT STGTEST; TYPE=2 is valid only for SYSEVENT ENQHOLD and ENQRLSE.

Default: TYPE=BLOCK`,ASID=asid`

Address space id

`,ASIDL=asidl`

Parameters

The parameters are explained as follows:

sysevent mnemonic

Identifies the SYSEVENT being requested. The valid options are:

DONTSWAP
 ENCASSOC
 ENCSTATE
 ENQHOLD
 ENQRLSE
 OKSWAP
 REQASCL
 REQASD
 REQFASD
 REQLPDAT
 REQSRMST
 STGTEST
 TRAXERPT
 TRAXFRPT
 TRAXRPT
 TRANSWAP

See “SYSEVENT Mnemonics” on page 172 for descriptions of these options.

,ENTRY=SVC

,ENTRY=BRANCH

Specifies the instruction used to pass control to SRM.

Only users who do not hold a lock can specify ENTRY=SVC.

SYSEVENT Macro

Branch entry is required when the caller holds a lock. It is also the only supported entry for TRAXERPT, TRAXFRPT, TRAXRPT, REQFASD, and QVS.

TYPE=BLOCK

TYPE=BYTE

Indicates whether SYSEVENT STGTEST is to return values that reflect either available central and expanded storage, **or** available expanded storage.

TYPE=BYTE requests central and expanded storage; TYPE=BLOCK requests expanded storage.

TYPE=2

Indicates whether SYSEVENT ENQHOLD and ENQRLSE passes a parameter list in register 1. The parameter list must be non-pageable and addressable via the caller's primary address space. To map the parameter list, use the IRAEVPL mapping macro described in *z/OS MVS Data Areas, Vol 2 (DCCB-ITZYRETC)*.

ASID=register

ASIDL=asid

Specifies the address space identifier in ASIDL=*asid*, or the register containing the address space identifier in ASID=*register*.

Either ASIDL or ASID is required for REQASD, REQFASD, ENQHOLD, and ENQRLSE.

SYSEVENT Mnemonics

A description of the SYSEVENTs available for use follows. These mnemonics are grouped according to the basic function that they perform.

Notify SRM of Transaction Completion

The SYSEVENTs TRAXRPT, TRAXFRPT, and TRAXERPT notify SRM that a subsystem transaction has completed and provide the transaction's starting time or elapsed time and, optionally, its resource utilization. This performance data can be reported using the resource management facility (RMF).

As of MVS/ESA SP 5.1 you should instead use the workload management services to notify SRM of transaction start and completion times, as well as notifying SRM of transaction delays encountered. For more information, see *z/OS MVS Programming: Workload Management Services*.

In addition to the general requirements for SYSEVENTs, TRAXRPT, TRAXFRPT, and TRAXERPT require the user to:

- Provide a parameter list
- If the issuing program is disabled, ensure that the parameter list and save area are fixed
- Provide error recovery

A description of the individual mnemonics follows:

TRAXRPT

Notifies SRM that a transaction has completed and provides its start time.

Register 1 must point to a serialized parameter list with the following format:

Offset in Hex	Length	Field Description
00	8	Transaction start time in store clock instruction (STCK) format
08	8	Subsystem name

Offset in Hex	Length	Field Description
10	8	Transaction name or blanks
18	8	User identification (USERID) or blanks
20	8	Transaction class or blanks

The names must be in EBCDIC format, left-justified, and padded with blanks. Note that the subsystem name is restricted to four characters in length even though the parameter list provides an eight-character field. Use the first four characters of the field for the subsystem name.

TRAXFRPT

Notifies SRM that a transaction has completed and provides the elapsed time. Because the issuer calculates the elapsed time before issuing the macro, this path is shorter than the path for TRAXRPT. Register 1 must point to a serialized parameter list with the following format:

Offset in Hex	Length	Field Description
00	4	Transaction elapsed time (1.024 milliseconds units)
04	4	Zero
08	8	Subsystem name
10	8	Transaction name or blanks
18	8	User identification (USERID) or blanks, as specified on the USERID parameter in the IEAICSxx parmlib member
20	8	Transaction class or blanks

Note: To map the parameter list, use the IHATRBPL mapping macro described in *z/OS MVS Data Areas, Vol 2 (DCCB-ITZYRETC)*.

The names must be in EBCDIC format, left-justified, and padded with blanks. Note that the subsystem name is restricted to four characters in length.

TRAXERPT

Notifies SRM that a transaction has completed, provides its start time, and includes resource utilization data for determining service consumption. Register 1 must point to a serialized parameter list in the following format:

Offset in Hex	Length	Field Description
00	8	Transaction start time in STCK format
08	8	Subsystem name
10	8	Transaction name or blanks
18	8	User identification (USERID) or blanks
20	8	Transaction class or blanks
28	8	Task (TCB) time in STCK format or zeros
30	8	SRB time in STCK format or zeros
38	8	Main storage occupancy in page seconds (pages times msec, where msec is task (TCB) time in 1.024 millisecond units)
40	4	Logical I/O count or zeros
44	1	X'00' if the previous field contains the logical I/O count X'80' if the previous field contains the device connect time interval (DCTI)
45	3	Reserved must be zero

SYSEVENT Macro

Note: To map the parameter list, use the IHATREPL mapping macro described in *z/OS MVS Data Areas, Vol 2 (DCCB-ITZYRETC)*.

The names must be in EBCDIC format, left-justified, and padded with blanks. Note that the subsystem name is restricted to four characters in length.

Return and Reason Codes

When processing is completed for TRAXRPT, TRAXFRPT, and TRAXERPT SYSEVENTs, the subsystem regains control at the instruction following the SYSEVENT macro. The last byte of register 15 contains one of the following hexadecimal return codes:

Table 26. Return Codes for the SYSEVENT Macro

Return Code	Meaning
00	Data for the transaction has been reported successfully to the SRM.
08	Processing could not be completed at this time. No queue elements are available for recording the data. No data is reported to the SRM, but an immediate reissue could be successful.
0C	Reporting is temporarily suspended for one of the following reasons: <ul style="list-style-type: none">• RMF workload activity reporting is not active.• There is no installation control specification (IEAICSxx parmlib member with RPGN specified for some subsystem other than TSO) in effect.• The TOD clock is stopped. No data is reported, but a later reissue could be successful.
10	Reporting is inoperative. The TOD clock is in error or the reporting interface is not installed. No data is reported.

Example 1

Use the SYSEVENT TRAXRPT to report transaction data providing transaction identifiers and the transaction start time. In the example, TRAXDESC is the name of a storage area that is initialized with the subsystem name, transaction name, userid and class information needed to pass to SRM.

```
.
.
.
Transaction begins      Initialize transaction identifiers
      (TRAXDESC)
      STCK INITTIME      Save start time
.
Process transaction
Transaction completes
      LA R13,SVAREA      Provide 72-byte save area
      LA R1,PARMS         Point to parameter area
      MVC 0(8,R1),INITTIME Move in start time
      MVC 8(32,R1),TRAXDESC Get subsystem name, transaction
                           name, userid, and class
      SYSEVENT TRAXRPT
.
.
INITTIME DS D
PARMS     DS 5D
SVAREA    DS 18F
TRAXDESC  DS CL40
```

Example 2

Use the SYSEVENT TRAXERPT to report transaction data, providing transaction identifiers, start time and resource utilization data. In the example, TRAXDESC is the name of a storage area that is initialized with the subsystem name, transaction name, userid and class information needed to pass to SRM.

```

.
.
Transaction begins      Initialize transaction identifiers
      (TRAXDESC)
STCK INITTIME          Save start time
.
Process transaction    Accumulate resource utilization data
      (TRAXDESC)
.
Transaction completes
LA  R13,SVAREA         Provide 72-byte save area
LA  R1,PARMS           Point to parameter area
MVC 0(8,R1),INITTIME   Move in start time
MVC 8(64,R1),TRAXDESC  Get subsystem name, transaction
                        name, user id, class, and
                        resource utilization data

SYSEVENT TRAXERPT
.
.
.
INITTIME DS D
PARMS    DS 9D
SVAREA   DS 18F
TRAXDESC DS CL72

```

Example 3

Use the SYSEVENT TRAXFRPT to report transaction data, providing transaction identifiers and calculating the elapsed time. In the example, TRAXDESC is the name of a storage area that is initialized with the subsystem name, transaction name, userid and class information needed to pass to SRM.

```

.
.
Transaction begins      Initialize transaction identifiers (TRAXDESC)
.
.
Process transaction     Calculate elapsed time (TOTLTIME)
.
Transaction completes   Calculate elapsed time (TOTLTIME)
LA  R13,SVAREA         Provide 72-byte save area
LA  R1,PARMS           Point to parameter area
MVC 0(4,R1),TOTLTIME    Move in elapsed time
XC  4(4,R1),4(R1)       Clear reserved field
MVC 8(32,R1),TRAXDESC  Get subsystem name, transaction name,
                        user id, and class

SYSEVENT TRAXFRPT
.
.
.
TOTLTIME DS F
PARMS    DS 5D
SVAREA   DS 18F
TRAXDESC DS CL40

```

Control Swapping

The SYSEVENTs DONTSWAP, OKSWAP, and TRANSWAP control swapping. The choice of mnemonic depends on the period of time for which the address space is to be non-swappable.

For a short period of time (less than one minute), use DONTSWAP to make it non-swappable and OKSWAP to make it swappable.

For an extended period of time (more than one minute), use TRANSWAP to make the address space non-swappable and OKSWAP to make it swappable.

Note: If you specify an ASID with DONTSWAP, OKSWAP, or TRANSWAP, that ASID must specify the home address space. In other words, you can only control swapping in the address space in which the SYSEVENT is issued. If you specify a different address space, the request will fail.

A description of the individual mnemonics follows:

DONTSWAP

Notifies SRM that the address space from which this SYSEVENT is issued cannot be swapped out until the system receives a matching OKSWAP for each DONTSWAP issued or until the jobstep ends.

No input parameters are required. One of the following codes will be returned in register 1, byte 3:

Hexadecimal Code	Meaning
00	The request was honored.
04	The request was not honored because it was not for the current address space.
08	The request was not honored because the issuer was not authorized or the outstanding count of DONTSWAP requests had reached its maximum.

OKSWAP

Notifies SRM that the address space from which the SYSEVENT was issued can be considered for swapping.

No input parameters are required. One of the following codes will be returned in register 1, byte 3:

Hexadecimal Code	Meaning
00	The request was honored.
04	The request was not honored because it was not for the current address space.
08	The request was not honored because the issuer was not authorized.

TRANSWAP

Forces a swap out. After the subsequent swap-in, frames are allocated from preferred storage and the address space is non-swappable. TRANSWAP prevents programs from allocating frames in reconfigurable storage. If the program issuing SYSEVENT depends on the transition to complete, you should

ensure that register 1 contains the address of an ECB. SYSEVENT will then post this ECB when it swaps out the address space. If no dependency exists, set register 1 to 0 (zero).

One of the following codes will be returned in register 1, byte 3:

Hexadecimal Code	Meaning
00	The request was honored. If an ECB was specified, your program should issue a WAIT macro specifying the same ECB.
04	The transition was previously done or the address space is permanently non-swappable. If an ECB was specified it will not be posted.

If an ECB was specified, the following POST codes may occur in the last three bytes of the ECB:

Hexadecimal Code	Meaning
000000	The transition is complete.
000004	The address space became non-swappable before it could be swapped out.

Example 1

Make the current address space non-swappable for a time period of less than one minute.

```
SYSEVENT DONTSWAP
      .
      .
      .
SYSEVENT OKSWAP
```

Example 2

Make the current address space non-swappable for an indefinite period of time.

```
SYSEVENT TRANSWAP
ST    1, RETCODE
CLI   TSWP_RC, 0
BNE   FAILED
WTO   'TSWP SUCCESSFUL'
      ...
B     DONE
FAILED EQU *
WTO   'TSWP UNSUCCESSFUL (BAD RC)'
DONE  EQU *
      ...
RETCODE DS 0F
        DS CL3
TSWP_RC DS FL1
```

Obtain System Measurement Information

STGTEST provides information about the current physical use of resources. This is not an indication of how much virtual storage your installation will allow you to obtain. For more information on obtaining virtual storage for hiperspaces or data spaces, see DSPSERV.

SYSEVENT Macro

The user must supply the address of a storage area large enough to store the requested data.

A description of the individual mnemonics follows:

STGTEST

Returns information about the amount of storage available in the system. The purpose of SYSEVENT STGTEST is to help an application decide whether to use an additional virtual storage area, such as a hiperspace. In ESA/390 mode, this information is about either central and expanded **or** only expanded storage.

In z/Architecture mode, this information is about central storage only. The TYPE=BLOCK parameter is not valid in this case, because expanded storage is not supported in z/Architecture mode. For compatibility, STGTEST TYPE=BLOCK will return the same data as TYPE=BYTE in z/Architecture mode. This ensures that applications that use STGTEST before creating a hiperspace will not fail.

When you use this information, be aware of the dynamic nature of storage.

Output of the SYSEVENT STGTEST represents the current state of storage and does not reserve this storage for the caller or guarantee that it will be available for use.

TYPE=BYTE

TYPE=BLOCK

In ESA/390 mode, this parameter specifies whether the system is to provide information about central storage and expanded storage (through TYPE=BYTE), or expanded storage (through TYPE=BLOCK). The default is TYPE=BLOCK.

In z/Architecture mode, TYPE=BLOCK is not valid because expanded storage is not supported. For compatibility, STGTEST TYPE=BLOCK and TYPE=BYTE return the same data in z/Architecture mode.

Register 1 must contain the address of a three-word output area where SRM is to return the information. After SRM returns, each word contains a storage amount that represents a specific number of frames. Before you choose a number to use as the basis for decision, be aware of how your decision affects the performance of the system. The meaning of the returned values is:

- Use of the first number will affect system performance very little, if at all.
- Use of the second number might affect system performance to some degree.
- Use of the third number might substantially affect system performance.

If you base decisions on the value in the second or third word, SRM may have to take processor storage away from other programs and replace it with auxiliary storage.

If the requesting address space has storage isolation

If you are running your system in workload management compatibility mode, you may be using storage isolation. To calculate the values for applications that have storage isolation, SRM first calculates the values for each word as if storage isolation were not in effect. It then modifies the values depending on:

- The number of frames the address space already has
- The minimum and (optionally) the maximum values specified through the PWSS keyword in the IEAIPSxx member of parmlib

The best way to understand SRM's calculations when storage isolation is in effect is through an example. Consider an application that issues SYSEVENT STGTEST,TYPE=BLOCK to find out the number of expanded storage frames available. In this example, the application:

- Is running in a performance group with storage isolation in effect
- **Currently holds 100 frames**

Assume that SRM has calculates a value of 20 frames for the first word if storage isolation were not in effect. In this example, 20 is the **base number**.

Table 27 shows how SRM adjusts the base number to arrive at the value for word one. Note how SRM takes into account the boundaries that are set on the PWSS keyword. In calculating a value, SRM might add to the base number to match the PWSS minimum number of frames. On the other hand, SRM might subtract from the base number to match the PWSS maximum number of frames.

Table 27. Example of Output of Word One from SYSEVENT STGTEST

Relationship between what the application holds (100 frames) and its PWSS setting	PWSS setting	What value does SRM return?
The application holds fewer frames than its PWSS minimum.	(190,*)	90 The PWSS minimum setting guarantees at least 190 frames. Therefore, SRM adds 70 to the base number to bring the application to its PWSS minimum.
The application holds more frames than its PWSS minimum, but less than its PWSS maximum minus the base number.	(50,*)	20 (the base number) The PWSS minimum setting guarantees at least 50 frames and the application already holds 100. Therefore, SRM does not make changes to the base number.
The application holds more frames than its PWSS minimum, and more than its PWSS maximum minus the base number.	(50,105)	5 If SRM returned 20 frames (the base number), the application would exceed its PWSS maximum. Therefore, SRM changes the base number to bring the application to its PWSS maximum.
The application holds more frames than its PWSS maximum.	(30,90)	Zero The application already exceeds its PWSS maximum; therefore, SRM returns the value 0.

SRM adjusts the values for word two for storage isolation in the same way it adjusts word one, as described in Table 27.

SRM does not directly consider storage isolation in calculating word three. However, because the value of word two determines the value of word three, the value SRM returns for word three indirectly reflects storage isolation.

If the example in Table 27 was for a TYPE=BYTE request, the effect of storage isolation would be exactly the same. However, because the base number would have reflected both expanded **and** central storage, the base number would probably have been greater than 20.

See *z/OS MVS Initialization and Tuning Guide* if you want more information about:

SYSEVENT Macro

- The syntax of the PWSS keyword in IEAIPSxx
- Storage isolation

Example

An application needs a standard hiperspace. Before it makes the request, the application uses SYSEVENT STGTEST to find out how much expanded storage is available. The values that SRM returns determine how large a hiperspace the application will create.

Obtain a report on the available expanded storage in the system.

```
LA 1,ESPARM
SYSEVENT STGTEST,TYPE=BLOCK
.
.
ESPARM DS 3F
```

The application will base its decision on the numbers in the first and second words of the output area.

Obtain Address Space Classification Information (REQASCL)

The REQASCL SYSEVENT provides information about an address space's classification information. You must specify the address space id (ASID) with either the ASID=*register* or the ASIDL=*asid* parameter.

The user must supply the address of a storage area large enough to hold the requested data.

A description of the individual mnemonic follows:

REQASCL

Returns classification information about an address space.

The name is in EBCDIC format, left-justified, and padded with blanks.

Input Register Information

Register 1 must point to a parameter list. The parameter list for REQASCL must be non-pageable and addressable via the caller's primary address space. To map the parameter list for REQASCL, use the IRARASC mapping macro described in *z/OS MVS Data Areas, Vol 2 (DCCB-ITZYRETC)*.

Return and Reason Codes

When processing is complete for the REQASCL, the last byte of register 15 contains one of the following hexadecimal return codes:

Table 28. Return Codes for REQASCL

Return Code	Meaning
00	Successful completion.
04	Classification information returned may not reflect how the address space is being managed
08	Input parameter list is not properly initialized (eyecatcher, version or size specified is too small)
12	Classification information is not available

Input Register Information

Register 1 must point to a parameter list, as mapped by the IRARASC macro.

Obtain Address Space Related Information (REQASD and REQFASD)

The SYSEVENTs REQASD and REQFASD provide information about an address space's workload activity. You must specify the address space id (ASID) with either the ASID=*register* or the ASIDL=*asid* parameter.

Both return the same kind of information. REQFASD is quicker; it does not serialize data collection, and does not provide recovery of its own. The user must provide the recovery for REQFASD.

The user must supply the address of a storage area large enough to hold the requested data.

A description of the individual mnemonics follows:

REQASD

Returns workload activity information about an address space.

REQFASD

Returns the same information as REQASD, but is a fast path SYSEVENT, with no recovery of its own.

The names are in EBCDIC format, left-justified, and padded with blanks.

Input Register Information

For both REQASD and REQFASD, register 1 must point to a parameter list. The parameter list for REQASD must be non-pageable and addressable via the caller's primary address space. The parameter list for REQFASD must be addressable via the caller's primary address space. To map the parameter list for both REQASD and REQFASD, use the IRARASD mapping macro described in *z/OS MVS Data Areas, Vol 2 (DCCB-ITZYRETC)*.

For REQFASD, register 13 must point to a workarea. The workarea must be addressable via the caller's primary address space. The workarea must be the length in the RQFASDWA field which is defined in the IRARASD mapping.

Return and Reason Codes

When processing is complete for the REQASD and REQFASD SYSEVENTs, the last byte of register 15 contains one of the following hexadecimal return codes:

Table 29. Return Codes for REQASD and REQFASD

Return Code	Meaning
00	Successful completion.
04	Processing could not be completed at this time. A mode switch or policy activation is in progress. A later reissue could be successful.
08	The parameter list is too small.
12	The ASID is not valid.

Obtain Workload Management Mode Status Information (REQSRMST)

The REQSRMST SYSEVENT allows a caller to obtain information about the state of workload management. It returns a parameter list that includes such things as:

- IPS and ICS, parmlib member suffixes (if in compatibility mode)
- OPT parmlib member suffix
- Active service policy information
- When and where the policy was activated
- Active service definition information
- When and where the service definition was installed

Note: The SRMSTCAP flag is provided to prospective callers of the REQLPDAT SYSEVENT, to test if that SYSEVENT is available on the system. On systems prior to z/OS V1R3, callers should first invoke the REQSRMST SYSEVENT and check the SRMSTCAP flag before invoking the REQLPDAT SYSEVENT. (See “Obtain Data for Defined Capacity (REQLPDAT)” for more information.)

Input Register Information

Register 1 must point to a parameter list. The parameter list must be non-pageable and addressable via the caller's primary address space. To map the parameter list, use the IRASRMST mapping macro described in *z/OS MVS Data Areas, Vol 2 (DCCB-ITZYRETC)*.

Return and Reason Codes

When processing is complete for the REQSRMST SYSEVENT, the last byte of register 15 contains one of the following hexadecimal return codes:

Table 30. Return Codes for REQSRMST

Return Code	Meaning
00	Successful completion.
08	The parameter list is too small.

Obtain Data for Defined Capacity (REQLPDAT)

The REQLPDAT SYSEVENT allows a caller to obtain performance data related to a defined capacity. When using a pricing model based on the logical capacity of LPARs (rather than on a CEC's physical capacity), the customer will specify the defined capacity for a partition. Software that runs on a logical partition are then charged for this defined capacity.

Monitoring products such as RMF need this performance data to better analyze the partition's average CPU consumption, how often WLM is capping the partition to enforce the defined capacity, the average weight of the partition, etc.

Notes:

1. The REQLPDAT SYSEVENT, introduced in z/OS V1R3, is available to downlevel systems. Before invoking this SYSEVENT on downlevel systems, however, you must first invoke the REQSRMST SYSEVENT. If the SRMSTCAP flag is turned on, then the system will support the REQLPDAT SYSEVENT. Otherwise, invoking it will result in an abnormal termination. (See “Obtain Workload Management Mode Status Information (REQSRMST)” for more information.)

2. You must set the field LPDatLen to the length of the entire parameter list before invoking the REQLPDAT SYSEVENT. If the size of the parameter list is not known (this will be the case when the SYSEVENT is invoked for the first time), then you should set LPDatLen to either 0 or else obtain a sufficiently large parameter list and then adjust LPDatLen accordingly. If the LPDatLen value is smaller than the size of the parameter list, then the SYSEVENT will fail with return code 4. In this case, the LPDatLen field will be set to the actual length of the parameter list.

Input Register Information

Register 1 must point to a parameter list. The parameter list must be non-pageable and addressable via the caller's primary address space. To map the parameter list, use the IRAMDC mapping macro described in *z/OS MVS Data Areas, Vol 2 (DCCB-ITZYRETC)*.

Return and Reason Codes

When processing is complete for the REQLPDAT SYSEVENT, the last byte of register 15 contains one of the following hexadecimal return codes:

Table 31. Return Codes for REQLPDAT

Return Code	Meaning
00	Successful completion.
04	The parameter list is too small.

Identify Holder of a Resource (ENQHOLD)

Use the ENQHOLD SYSEVENT to identify a holder of a resource causing contention. SRM may boost the service to the holder of the resource to help resolve the contention more quickly. A holder can be either an address space or an enclave. You must specify the address space in the ASID, or ASIDL parameter. If you want to specify that an enclave is holding the resource, you must specify x'8000' in the ASID parameter, and access registers (AR) 0 and 1 must contain the enclave token. The enclave token must have been obtained from the IWMECREA macro.

If you issue the ENQHOLD SYSEVENT, you are responsible for issuing a corresponding ENQRLSE SYSEVENT when the holder has released the resource.

There are some considerations to be aware of when using enclaves for tasks that serialize on resources using the ENQ macro or the latch manager callable services. A task cannot change its transaction status, that is, cannot join or leave an enclave, while holding a resource using ENQ or the latch manager. Otherwise, enqueue promotion processing may not work properly. The recommended sequence is:

- Join an enclave (through IWMEJOIN, IWMSTBGN, or SYSEVENT ENCASSOC).
- Obtain resource with ENQ or latch manager.
- Process using serialized resource.
- Release resource.
- Leave an enclave (through IWMELEAV, IWMSTEND, or SYSEVENT ENCASSOC).

In addition, to ensure correct enqueue promotion processing, a task executing in an enclave should **not** make the following types of ENQ requests:

SYSEVENT Macro

- directed enqueues, that is, issuing the ENQ macro with the TCB= parameter
- matching task enqueues, that is, issuing the ENQ macro with the MTCB or MASID parameter.

Input Register Information

If you are specifying an enclave as a holder, access register (AR) 0 and 1 must contain an enclave token.

If this SYSEVENT is invoked with the TYPE=2 keyword, then register 1 must point to a parameter list. The parameter list must be non-pageable and addressable via the caller's primary address space. To map the parameter list, use the IRAEVPL mapping macro described in *z/OS MVS Data Areas, Vol 2 (DCCB-ITZYRETC)*.

Return and Reason Codes

When processing is complete for the ENQHOLD SYSEVENT, the last byte of register 15 contains one of the following hexadecimal return codes:

Table 32. Return Codes for ENQHOLD

Return Code	Meaning
00	Successful completion.
08	Invalid enclave token specified.

Identify that a Holder has Released Resource (ENQRLSE)

Use the ENQRLSE sysevent to notify SRM that the holder of a resource causing contention has released the resource. The inputs must be the same as those for the ENQHOLD SYSEVENT previously issued for the holder. See the description of ENQHOLD for considerations related to using enclaves for tasks that serialize resources.

Input Register Information

If you are specifying an enclave as a holder, access register (AR) 0 and 1 must contain an enclave token.

If this SYSEVENT is invoked with the TYPE=2 keyword, then register 1 must point to a parameter list. The parameter list must be non-pageable and addressable via the caller's primary address space. To map the parameter list, use the IRAEVPL mapping macro described in *z/OS MVS Data Areas, Vol 2 (DCCB-ITZYRETC)*.

Return and Reason Codes

When processing is complete for the ENQRLSE SYSEVENT, the last byte of register 15 contains one of the following hexadecimal return codes:

Table 33. Return Codes for ENQRLSE

Return Code	Meaning
00	Successful completion.
08	Enclave token is invalid.

Associate an Enclave with an Address Space (ENCASSOC)

Use the ENCASSOC SYSEVENT to associate an enclave and an address space so that they are related for purposes of storage management.

Input Register Information

Register 1 must point to a parameter list. The parameter list must be non-pageable and addressable via the caller's primary address space. To map the parameter list, use the IRAEVPL mapping macro described in *z/OS MVS Data Areas, Vol 2 (DCCB-ITZYRETC)*.

Set the State for an Enclave (ENCSTATE)

Use the ENCSTATE SYSEVENT to set the state of an enclave to either idle or non-idle. You must specify x'8000' in the ASID parameter, and access registers (AR) 0 and 1 must contain the enclave token. The enclave token must have been obtained from the IWMECREA macro.

Input Register Information

Register 1 must point to a parameter list. The parameter list must be non-pageable and addressable via the caller's primary address space. To map the parameter list, use the IRAEVPL mapping macro described in *z/OS MVS Data Areas, Vol 2 (DCCB-ITZYRETC)*.

Access register (AR) 0 and 1 must contain the enclave token whose state you are setting.

Query Amount of Free AUX Storage (FREEAUX)

Use the FREEAUX SYSEVENT to receive the number of free AUX storage slots that can safely be used without causing an AUX shortage.

Output Register Information

Register 0 will contain the recommended number of free AUX storage slots.

Query a Virtual Server (QVS)

Products can use the query virtual server (QVS) interface to obtain a virtual server's ID and capacity. A virtual server is the "logical hardware" environment in which an image runs. In the case of an image running in a logical partition, the virtual server is the logical partition. In the case of an image running as a first level VM guest, the virtual server is the first level guest. Note that the virtual server concept is not extended to second or higher level VM guests. In those cases, the virtual server is still considered to be the first level VM guest. For an image running in basic mode, the virtual server is the whole machine. In other words, for a CPC in basic mode, the logical hardware environment equals the physical hardware environment.

On a zSeries machine, customers can define a capacity limit for each logical partition. This limit is enforced by Workload Manager based on the average CPU usage of the logical partition. Peaks of CPU usage are allowed above the limit as long as the average CPU usage stays below the limit.

This service gives software vendors or software providers the option of licensing their software to a virtual server, and a provisional method of providing workload pricing until their products can be fully License Manager-enabled.

One way a software product can use the virtual server ID and capacity query service is to call the service when it starts. Based on the information returned, the product can verify that it is licensed to run on the specific virtual server and that the

SYSEVENT Macro

virtual server does not have more capacity than the product is licensed for. Note that this service does *not* make any of these licensing checks — it simply returns the information to enable a software product to make the appropriate checks based on the conditions of its license.

Along with this SYSEVENT, there is also a C interface, IWMQVS. Both forms of this query return a QVS structure which maps the returned identification and capacity information. The assembler mapping is provided by the macro IRAQVS, and the C/C++ mapping is provided in IWMQVS.H. Before calling this service, the caller must provide storage for the QVS structure and set the field QvsLen to the length of the structure. On return, the caller can look at fields QvsVer and QvsFlags to determine which fields have been filled in. QvsFlags contains three flags:

QvsCecValid

The physical hardware level information is valid.

QvsImgValid

The logical partition level information is valid. This flag will be off if not running in logical partition mode.

QvsVmValid

The virtual machine information is valid. This flag will be off if not running in a virtual machine.

The physical hardware level information is provided in the following fields:

- QvsCecManufacturerName
- QvsCecPlantofManufacture
- QvsCecMachineType
- QvsCecModelId
- QvsCecSequenceCode
- QvsCecCapacity

The logical partition level information is provided in the following fields:

- QvsImgLogicalPartitionId
- QvsImgLogicalPartitionName
- QvsImgCapacity

The virtual machine information is provided in the following fields:

- QvsVmName
- QvsVmCapacity

QvsCecCapacity, QvsImgCapacity, and QvsVmCapacity contain the maximum service rate that theoretically could be achieved at each level. The value is in millions of service units per hour (MSU).

QvsCecCapacity is equal to the individual CPU speed multiplied by the number of online and offline physical CPUs.

If QvsImgValid is on, the image is in ESAME mode, and QvsVmValid is off, then QvsImgCapacity is equal to one of the following:

- The partition's defined capacity set via the Hardware Management Console, if any
- The individual CPU speed multiplied by the number of online and offline logical CPUs, if the partition is uncapped and has no defined capacity
- The capacity at the partition's weight, if the partition is capped via the Hardware Management Console.

If QvslmgValid is on, and either the image is in ESA/390 mode or QvsVmValid is on, then QvslmgCapacity is equal to the individual CPU speed multiplied by the number of online and offline logical CPUs.

QvsVmCapacity is the individual CPU speed multiplied by the number of online and offline virtual CPUs.

In all cases, the individual CPU speed is based on the MP factor for the number of online and offline physical CPUs.

Note that the capacity of a virtual server can change dynamically. One example of a dynamic capacity change is a CPU upgrade on demand of the underlying hardware. A second example is a dynamic change of the defined capacity limit for a logical partition. If an unauthorized program is interested in knowing about dynamic capacity changes, it must poll the virtual server ID and capacity query service. Given dynamic capacity changes are rare, a low polling rate should be sufficient.

Authorized programs interested in knowing about dynamic capacity changes can also listen for ENF signal 61. This ENF is signaled when a change in dynamic capacity occurs and provides the listener with the new capacity at each level in the hierarchy.

Return and Reason Codes

When processing is complete for the QVS SYSEVENT, the last byte of register 15 contains one of the following hexadecimal return codes:

Table 34. Return Codes for QVS

Return Code	Meaning
00	Successful completion.
04	The parameter list is too small.

SYSEVENT Macro

TCBTOKEN — Request or Translate the TTOKEN

Description

The TTOKEN is the 16-byte identifier of a task. Unlike a TCB address, each TTOKEN is unique within the IPL; the system does not reassign this same identifier to any other TCB.

The TCBTOKEN macro provides five mutually exclusive services depending on how you specify the TYPE parameter:

- TYPE=TOTOKEN gives you the TTOKEN for the task associated with a specified TCB address.
- TYPE=TOTCB gives you the TCB address for a specified TTOKEN.
- TYPE=CURRENT gives you the TTOKEN for the current task.
- TYPE=PARENT gives you the TTOKEN for the task that attached the current task.
- TYPE=JOBSTEP gives you the TTOKEN for the job step task.

Typical situations when you would use TYPE=TOTOKEN are:

- When you create a data space and want to assign ownership of the data space to a second task.

In this case, you know the TCB address for the second task, but you don't know its TTOKEN (for input to the DSPSERV CREATE macro). Use TYPE=TOTOKEN to obtain the TTOKEN.

- When you want to delete a data space you do not own.

In this case, you know the TCB address for the other task, but you don't know its TTOKEN (for input to the DSPSERV DELETE macro). Use TYPE=TOTOKEN to obtain the TTOKEN.

- When you want to know whether the owner of a data space still exists.

In this case, you know the TTOKEN for the owning task. If the system returns the TCB address in response to the TYPE=TOTCB parameter, the task still exists.

z/OS MVS Programming: Extended Addressability Guide describes STOKENs and TTOKENs.

Environment

The requirements for the caller are:

Minimum authorization:	Problem or supervisor state, any PSW key
Dispatchable unit mode:	For TOTOKEN or TOTCB requests, the caller can be in task or SRB mode. For CURRENT, PARENT, and JOBSTEP requests, the caller must be in task mode.
Cross memory mode:	Any
AMODE:	31-bit
ASC mode:	Primary or AR
Interrupt Status:	Enabled or disabled for I/O and external interrupts
Locks:	For TOTOKEN and TOTCB requests, the caller must hold the local lock or CML lock of the specified address space. For CURRENT, PARENT, and JOBSTEP requests, there is no requirement.
Control parameters:	Can reside in the primary address space or in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL)

TCBTOKEN Macro

Programming Requirements

None.

Restrictions

None.

Register Information

After the caller issues the macro, the macro might use some registers as work registers or might change the contents of some registers. When the macro returns control to the caller, the contents of these registers are not the same as they were before the macro was issued. Therefore, if the caller depends on these registers containing the same value before and after issuing the macro, the caller must save these registers before issuing the macro and restore them after the system returns control.

When control returns to the caller, the general purpose registers (GPRs) contain:

Register	Contents
0-1	Used as work registers by the macro
2-13	Unchanged
14	Used as a work register by the macro
15	Return code

When control returns to the caller, the access registers (ARs) contain:

Register	Contents
0-1	Used as work registers by the macro
2-13	Unchanged
14-15	Used as work registers by the macro

Performance Implications

None.

Syntax

The standard form of the TCBTOKEN macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
<i>b</i>	One or more blanks must precede TCBTOKEN.
TCBTOKEN	
<i>b</i>	One or more blanks must follow TCBTOKEN.

TYPE=TOTTOKEN	Note: See the table following this diagram for information on parameter usage with TYPE.
TYPE=TOTCB	

TYPE=CURRENT
 TYPE=PARENT
 TYPE=JOBSTEP

,TCB=*tcb addr* *tcb addr*: RX-type address or register (2) - (12).

,TTOKEN=*ttoken addr* *ttoken addr*: RX-type address.

,ASCB=*ascb addr* *ascb addr*: RX-type address or register (2) - (12).
 ,STOKEN=*stoken addr* *stoken addr*: RX-type address.
Default: Home address space.

,RELATED=*value* *value*: Any valid macro parameter specification.

The following table shows how the parameters may be specified with the TYPE keywords.

Parameters	TYPE= TOTOKEN	TYPE= TOTCB	TYPE= CURRENT	TYPE= PARENT	TYPE= JOBSTEP
TCB	required	required	not valid	not valid	not valid
TTOKEN	required	required	required	required	required
ASCB	optional	optional	not valid	not valid	not valid
STOKEN	optional	not valid	not valid	not valid	not valid
RELATED	optional	optional	optional	optional	optional

Parameters

The parameters are explained as follows:

TYPE=TOTOKEN
TYPE=TOTCB
TYPE=CURRENT
TYPE=PARENT
TYPE=JOBSTEP

Specifies the type of TCB information requested, as follows:

TOTOKEN The system returns the TTOKEN of the task whose TCB address is specified in the TCB parameter. The TTOKEN is returned at the address specified by the TTOKEN parameter.

TOTCB The system returns the TCB address for the task whose TTOKEN is specified in the TTOKEN parameter. The TCB address is returned at the address specified by the TCB parameter.

CURRENT The system returns the TTOKEN of the currently active task. The TTOKEN is returned at the address specified by the TTOKEN parameter.

PARENT The system returns the TTOKEN of the task that attached the currently active task. The TTOKEN is returned at the address specified by the TTOKEN parameter.

JOBSTEP The system returns the TTOKEN of the job step task for the

TCBTOKEN Macro

address space in which the currently active task is running. The TTOKEN is returned at the address specified by the TTOKEN parameter.

,TCB=*tcb addr*

Specifies the TCB address. For TYPE=TOTTOKEN, *tcb addr* contains the TCB address that is to be translated to a TTOKEN. For TYPE=TOTCB, *tcb addr* points to a fullword where the system returns the TCB address for the task whose TTOKEN is specified by the TTOKEN parameter.

,TTOKEN=*ttoken addr*

Specifies the address of the 16-byte TTOKEN. For TYPE=TOTTOKEN, TYPE=CURRENT, TYPE=PARENT, and TYPE=JOBSTEP, *ttoken addr* is the address at which the TTOKEN associated with the specified TCB is returned. For TYPE=TOTCB, *ttoken addr* is the address of the TTOKEN for the task whose TCB address is to be obtained.

,ASCB=*ascb addr*

,STOKEN=*stoken addr*

Identifies the address space of the TCB. ASCB specifies the address of the fullword containing the ASCB address. STOKEN specifies the address of the 8-byte STOKEN that identifies the address space in which the TCB resides. If you do not specify either ASCB or STOKEN, TCBTOKEN uses the home address space by default.

,RELATED=*value*

Specifies information used to self-document macros by “relating” functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user and may be any valid coding values.

Abend Codes

None.

Return Codes

When TCBTOKEN returns control, register 15 contains one of the following hexadecimal return codes:

Table 35. Return Codes for the TCBTOKEN Macro

Return Code	Meaning and Action
00	Meaning: TCBTOKEN services completed successfully. Action: None.
04	Meaning: The input STOKEN or TTOKEN does not represent a valid address space. Action: Ensure that you specify a valid token on the STOKEN or TTOKEN keywords.
08	Meaning: No local lock was held. Action: Obtain the local lock before issuing TCBTOKEN.
0C	Meaning: A local lock was held, but not the local lock of the associated address space. Action: Obtain the correct local lock before issuing TCBTOKEN.
10	Meaning: The TCB could not be referenced. Action: Ensure that the input TCB address specified on the TCB keyword is valid.

Table 35. Return Codes for the TCBTOKEN Macro (continued)

Return Code	Meaning and Action
14	Meaning: The TCB did not pass the acronym check. Action: Ensure that the input TCB address specified on the TCB keyword is valid.
18	Meaning: The TCB has ended. Action: None required.
1C	Meaning: The TCB associated with the TTOKEN represents a different task than when the TTOKEN was obtained. Action: None required.
20	Meaning: An unexpected error occurred. Action: Reissue the TCBTOKEN macro.
24	Meaning: The contents of access register 1, used to address the parameter list, were not valid. Action: Either change your program to run in primary mode or set access register 1 to zero.
28	Meaning: The parameter list is not valid. Action: Ensure that the parameter list address is valid and addressable in the calling program's key.
2C	Meaning: The ASCB address is the address of the wait ASCB. The system cannot obtain the TTOKEN. Action: Specify an ASCB address which is not the wait ASCB.
30	Meaning: The task is scheduled for termination, but has not yet terminated. Action: None required.
34	Meaning: The caller is not running in task mode. This return code is valid only for TYPE=CURRENT, TYPE=PARENT, or TYPE=JOBSTEP. Action: Change your program to run in task mode.

Note: Return codes 04, 08, 0C, 1C, and 2C are valid only with TYPE=TOTTOKEN and TYPE=TOTCB.

Example 1

Obtain the TTOKEN for the task whose TCB address is specified in THEIR_TCB. The task resides in the address space whose ASCB address is specified in register 4. Store the returned TTOKEN in THEIR_TTOKEN.

```
TCBTOKEN  TYPE=TOTTOKEN,TCB=THEIR_TCB,TTOKEN=THEIR_TTOKEN,ASCB=(4)
```

Example 2

Obtain the TTOKEN for the currently active task and store it in CURRENT_TTOKEN.

```
TCBTOKEN  TYPE=CURRENT,TTOKEN=CURRENT_TTOKEN
```

Example 3

Obtain the TCB address of the job step TCB and store it in JOBSTEP_TCB_ADDR.

```
TCBTOKEN  TYPE=JOBSTEP,TTOKEN=JOBSTEP_TTOKEN
TCBTOKEN  TYPE=TOTCB,TTOKEN=JOBSTEP_TTOKEN,TCB=JOBSTEP_TCB_ADDR
```

TCBTOKEN—List Form

Use the list form of the TCBTOKEN macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage that the execute form of the macro uses to store the parameters.

Syntax

The list form of the TCBTOKEN macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede TCBTOKEN.
TCBTOKEN	
b	One or more blanks must follow TCBTOKEN.
<hr/>	
,RELATED=value	<i>value</i> : Any valid macro parameter specification.
,MF=L	

Parameters

The parameters are explained below:

,RELATED=value

Specifies information used to self-document macros by “relating” functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user and may be any valid coding values.

,MF=L

Specifies the list form of the TCBTOKEN macro.

TCBTOKEN—Execute Form

Use the execute form of the TCBTOKEN macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

Syntax

The execute form of the TCBTOKEN macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
-------------	--

One or more blanks must precede TCBTOKEN.

TCBTOKEN

One or more blanks must follow TCBTOKEN.

TYPE=TOTTOKEN

Note: See the table following this diagram for information on parameter usage with TYPE.

TYPE=TOTCB

TYPE=CURRENT

TYPE=PARENT

TYPE=JOBSTEP

,TCB=*tcb addr*

tcb addr: RX-type address or register (2) - (12).

,TTOKEN=*ttoken addr*

ttoken addr: RX-type address.

,ASCB=*ascb addr*

ascb addr: RX-type address or register (2) - (12).

,STOKEN=*stoken addr*

stoken addr: RX-type address.

Default: Home address space.

,RELATED=*value*

value: Any valid macro parameter specification.

,MF=(E,*cntl addr*)

cntl addr: RX-type address or register (1) - (12).

Parameters

The parameters are the same as those for the standard form of the TCBTOKEN macro with the following addition:

,MF=(E,*cntl addr*)

Specifies the execute form of the TCBTOKEN macro. This form uses a remote parameter list. The *cntl addr* specifies the address of the remote parameter list that the list form of the macro generates.

TCBTOKEN Macro

TCTL — Transfer Control from an SRB Routine

Description

The TCTL (transfer control) macro allows an SRB routine to exit from its processing and to pass control directly to a task.

Environment

Requirements for the caller are:

Minimum authorization:	Supervisor state and any PSW key
Dispatchable unit mode:	SRB
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O or external interrupts
Locks:	None held
Control parameters:	Must be in the caller's primary address space

Programming Requirements

The caller must include the following mapping macros:

- IHAPSA
- CVT with DSECT=YES

Restrictions

None.

Input Register Information

If you are using the default for the TCB parameter, on input to the TCTL macro, general purpose register (GPR) 4 must contain the address of the TCB.

Output Register Information

The system does not return to the caller after invoking this macro, so register contents on exit from the macro are not applicable.

Performance Implications

None.

Syntax

The TCTL macro is coded as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede TCTL.
TCTL	
b	One or more blanks must follow TCTL.

TCTL Macro

TCB=(4)

TCB=*tcbaddr*

Default: Register 4 contains TCB address.

tcbaddr: A-type address or registers (2) - (12).

Parameters

The parameters are explained as follows:

TCB=(4)

TCB=*tcbaddr*

Specifies the task designated for dispatching. Register 4 is the default; if you use the default, you must ensure that register 4 contains the appropriate TCB address.

Note: The TCB resides in storage below 16 megabytes.

ABEND Codes

070

See *z/OS MVS System Codes* for an explanation and programmer responses for this code.

Return and Reason Codes

The system does not return to the caller after this macro has been invoked, so return codes from the macro are not applicable.

Example

From SRB mode processing, terminate the SRB and give control to the task specified in register 4.

TCTL TCB=(4)

TESTAUTH — Test Authorization of Caller

Description

The TESTAUTH macro is used on behalf of a privileged or sensitive function to verify that its caller is appropriately authorized.

TESTAUTH supports the authorized program facility (APF) - a facility that permits the identification of programs that are authorized to use restricted functions. In addition, TESTAUTH provides the capability for testing for system key 0-7 and supervisor state. An EUT FRR may not be in force for a caller using BRANCH=NO.

Environment

The requirements for the caller are:

BRANCH=NO entry

Minimum authorization:	Problem or supervisor state, any key
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
Asc mode:	Primary
Interrupt status:	No requirement
Locks:	No requirement
Control Parameters:	Must be in primary address space

Programming Requirements

None.

Input Register Information

Before issuing the TESTAUTH macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter or using it as a base register.

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14	Used as a work register by the system
15	Return code

When control returns to the caller, the ARs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

TESTAUTH Macro

BRANCH=YES entry

Minimum authorization:	Supervisor state, any key
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
Asc mode:	Primary
Interrupt status:	No requirement
Locks:	Local lock held
Control Parameters:	Must be in primary address space

Programming Requirements

Callers must include the CVT mapping macro.

Input Register Information

Before issuing the TESTAUTH macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter or using it as a base register.

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0-3	Used as work registers by the system
4-13	Unchanged
14	Used as a work register by the system
15	Return Code

When control returns to the caller, the ARs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

Performance Implications

None.

Syntax

The TESTAUTH macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
␣	One or more blanks must precede TESTAUTH.
TESTAUTH	
␣	One or more blanks must follow TESTAUTH.

FCTN= <i>fctn</i>	<i>fctn</i> : Decimal digit 0 or 1 or register (2) - (12). Default: FCTN=0 if STATE or KEY is specified. Otherwise, the default is FCTN=1.
,STATE=NO ,STATE=YES	Default: STATE=NO
,KEY=NO ,KEY=YES	Default: KEY=NO
,RBLEVEL=2 ,RBLEVEL=1	Default: RBLEVEL=2 RBLEVEL is used only if KEY and/or STATE are specified; otherwise RBLEVEL is ignored.
,BRANCH=NO ,BRANCH=YES	Default: BRANCH=NO

Parameters

The parameters are explained as follows:

FCTN=*fctn*

Specifies the authorization of a program to be checked through APF.

FCTN=0 specifies that APF-authorization is not checked.

FCTN=1 specifies that APF-authorization is checked.

,STATE=NO

,STATE=YES

Specifies whether or not (YES or NO) a check is to be made for supervisor/problem program state. (Supervisor state is authorized, problem program state is not authorized.)

,KEY=NO

,KEY=YES

Specifies whether or not (YES or NO) a check is to be made of the protection keys. (Protection keys 0-7 are authorized, protection keys 8-15 are not authorized.)

Note: TESTAUTH is used to test one or more of three conditions: FCTN, STATE, or KEY. If any of the requested conditions are tested favorably, a return code of zero is returned in register 15. If all of the requested conditions are tested unfavorably, the return code is set to 4.

,RBLEVEL=2

,RBLEVEL=1

Specifies whether the TESTAUTH caller is a type 2, 3, or 4 SVC (RBLEVEL=2) or a type 1 SVC (RBLEVEL=1). If the TESTAUTH caller is not an SVC, specify RBLEVEL=1. Specify RBLEVEL only if you also specify KEY and/or STATE; otherwise RBLEVEL is ignored.

,BRANCH=NO

TESTAUTH Macro

,BRANCH=YES

Specifies a branch entry (YES) or an SVC entry (NO). If BRANCH=YES is specified, registers 2 and 3 are modified by the TESTAUTH routine. Only SVC routines can use BRANCH=YES.

ABEND Codes

None.

Return Codes

When control is returned, register 15 contains one of the following hexadecimal return codes:

Table 36. Return Codes for the SAMPLE Macro

Return Code	Meaning and Action
00	Meaning: Task is authorized. Action: None.
04	Meaning: Task is not authorized. Action: None.

Example 1

Test jobstep for APF authorization.

```
TESTAUTH FCTN=1
```

Example 2

Test for APF authorization and supervisor state and do not check protection keys.

```
TESTAUTH STATE=YES,KEY=NO,FCTN=1
```

TIMEUSED — Obtain Accumulated CPU or Vector Time

Description

The TIMEUSED macro returns an 8-byte hexadecimal number in a doubleword storage area that you specify. The number is the total CPU or vector time used by the current TCB or SRB up until you issue the macro. The format of the number is time-of-day (TOD) clock or microseconds time format.

If you use the SRBSTAT save and restore services, the number includes the interval between dispatch and save and between restore and TIMEUSED. It does not include the interval between save and restore. If you have not yet issued restore, the number includes only the interval between save and TIMEUSED.

TIMEUSED is also documented in the *z/OS MVS Programming: Assembler Services Reference ABE-HSP*, but without the LINKAGE=BRANCH parameter. That parameter is available only to authorized callers.

Environment

The requirements for the caller are:

Minimum authorization:	Supervisor state and PSW key 0 when you specify LINKAGE=BRANCH. Supervisor or problem state and any PSW key when you specify LINKAGE=SYSTEM.
Dispatchable unit mode:	Task or SRB when LINKAGE=BRANCH. Task when LINKAGE=SYSTEM.
Cross memory mode:	PASN=HASN=SASN or PASN≠HASN≠SASN
AMODE:	31- or 64-bit
ASC mode:	Primary or secondary when LINKAGE=BRANCH. Primary or AR (access register) when LINKAGE=SYSTEM.
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in the primary address space or be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL)

Programming Requirements

For information about programs in 64-bit addressing mode (AMODE 64), see *z/OS MVS Programming: Extended Addressability Guide*.

Restrictions

None.

Input Register Information

Register 13 must contain the address of an 18-word save area when you specify LINKAGE=BRANCH. You can provide the address through the use of standard linkage conventions.

Output Register Information

After the caller issues the macro, the macro might use some registers as work registers or might change the contents of some registers. When the macro returns control to the caller, the contents of these registers are not the same as they were

TIMEUSED Macro

before the macro was issued. Therefore, if the caller depends on these registers containing the same value before and after issuing the macro, the caller must save these registers before issuing the macro and restore them after the system returns control.

When control returns to the caller, the GPRs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14	Used as a work register by the macro
15	Return code

Performance Implications

None.

Syntax

The TIMEUSED macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
␣	One or more blanks must precede TIMEUSED.
TIMEUSED	
␣	One or more blanks must follow TIMEUSED.

STORADR= <i>addr</i>	<i>addr</i> : RX-type address or register (2)-(12).
,LINKAGE=SYSTEM ,LINKAGE=BRANCH	Default: LINKAGE=BRANCH
,RELATED= <i>value</i>	<i>value</i> : Any valid macro parameter specification
,CPU=TOD ,CPU=MIC ,VECTOR=TOD ,VECTOR=MIC	Default: CPU=TOD

Parameters

The parameters are explained as follows:

STORADR=*addr*

Specifies the 31-bit address of a doubleword area where the accumulated CPU or vector time is returned. The time interval is represented as an unsigned 64-bit binary number. If you specify CPU=TOD or VECTOR=TOD, bit 51 is the

low-order bit of the interval value and equivalent to 1 microsecond. If you specify CPU=MIC or VECTOR=MIC, bit 63 is the low-order bit of the interval value and equivalent to 1 microsecond.

,LINKAGE=SYSTEM

,LINKAGE=BRANCH

Specifies the type of linkage used in TIMEUSED processing.

LINKAGE=BRANCH indicates branch entry. You may specify or default to LINKAGE=BRANCH if you are a key zero supervisor state program running under a TCB or SRB. LINKAGE=SYSTEM indicates the linkage is by nonbranch entry.

,RELATED=value

Specifies information used to self-document macros by “relating” functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user and may be any valid coding values.

,CPU=TOD

,CPU=MIC

,VECTOR=TOD

,VECTOR=MIC

Specifies that TIMEUSED should return the total CPU or vector time in either TOD clock format (CPU=TOD or VECTOR=TOD) or in microseconds (CPU=MIC or VECTOR=MIC). You may specify CPU=MIC or VECTOR only if LINKAGE=SYSTEM.

Return Codes

When control returns to the caller, GPR 15 contains one of the following hexadecimal return codes.

Table 37. Return Codes for the TIMEUSED Macro

Return Code	Meaning and Action
00	Meaning: The service completed successfully. Action: None.
08	Meaning: Unexpected error Action: Retry the request.

Example 1

Using the unauthorized TIMEUSED service, request the total CPU time in TOD clock format to be stored at the address in register 2.

```
TIMEUSED    STORADR=(2),LINKAGE=SYSTEM,CPU=TOD
```

Example 2

Using the unauthorized TIMEUSED service in task mode, request the total vector time in microseconds to be stored at the address in register 2.

```
TIMEUSED    STORADR=(2),LINKAGE=SYSTEM,VECTOR=MIC
```

Example 3

Using the authorized TIMEUSED service, request the total CPU time in TOD clock format to be stored at the address in register 2.

```
TIMEUSED    STORADR=(2),LINKAGE=BRANCH
```

TIMEUSED Macro

T6EXIT — Type 6 Exit

Description

The T6EXIT macro returns control from a type 6 SVC. This exit macro can only be used in a type 6 SVC routine.

Environment

The requirements for the caller are:

Minimum authorization:	Supervisor state and PSW key 0
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Disabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	None

Programming Requirements

The caller must include the CVT mapping macro with DSECT=YES specified.

Restrictions

None.

Input Register Information

On input, general purpose register (GPR) 1 must point to a service request block (SRB) if RETURN=SRB is specified.

Output Register Information

For RETURN=CALLER, registers 0, 1, and 15 are returned from the type 6 SVC routine to the calling program (the issuer of the SVC). For RETURN=DISPATCH and RETURN=SRB, no registers are returned to the calling program.

Performance Implications

None.

Syntax

The T6EXIT macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede T6EXIT.
T6EXIT	
b	One or more blanks must follow T6EXIT.

T6EXIT Macro

RETURN=CALLER

Default: RETURN=CALLER

RETURN=DISPATCH

RETURN=SRB

Parameter

The explanation of the RETURN parameter is as follows:

RETURN

Specifies how the type 6 SVC has chosen to exit, which is one of the following:

- CALLER specifies that the return is directly to the caller or issuer of the SVC. The contents of GPRs 0, 1, and 15 at the time of the T6EXIT are returned to the issuer of the SVC. CALLER is the default return option.
- DISPATCH specifies that the return should be to the system to dispatch other work. This function is for the use of routines that have suspended the current task. When the task resumes, the issuer of the type 6 SVC receives control at the instruction after the SVC.
- SRB specifies that the system should immediately dispatch an SRB. This SRB must:
 - Be initialized by the type 6 SVC.
 - Be pointed to by register 1.
 - Run in the same address space as the SVC. The SRB has the same format as an SRB dispatched through the SCHEDULE macro.

ABEND Codes

None.

Return and Reason Codes

None.

Example

Terminate type 6 SVC processing and return control from the type 6 SVC to the caller of the SVC.

```
T6EXIT RETURN=CALLER
```

UCBINFO — Return Information from a UCB

Description

Use the UCBINFO macro to obtain information from a unit control block (UCB) for a specified device. The UCBINFO macro provides the following options:

DEVCOUNT	Returns a count of the UCBs for a device class or device group.
DEVINFO	Returns information about a device, specifically, why the device is offline. For the base UCB of a &pav., DEVINFO returns the number of alias UCBs that are defined and the number that are usable.
PATHINFO	Returns information about the device path and type of channel path associated with the device.
PATHMAP	Returns information about the device path.
PRFXDATA	Obtains a copy of the UCB prefix extension segment.
PAVINFO	Returns information about the alias UCBs for a parallel access volume.

The options of the UCBINFO macro have the same environmental specifications, programming requirements, restrictions, register information, and performance implications described below, except where noted in the explanations of each option.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state and any PSW key. For LINKAGE=BRANCH, all of the following: <ul style="list-style-type: none">• Supervisor state with key 0• 31-bit addressing mode• Primary ASC mode• Parameter list and any data areas it points to must be in fixed storage or, if the caller is disabled, in disabled reference (DREF) storage.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	The caller may hold locks, but is not required to hold any
Control parameters:	Control parameters must be in the primary address space or, for AR-mode callers, must be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming Requirements

Before issuing the UCBINFO macro, you can issue the UCBSCAN macro to obtain either the UCBPTR or the device number, which you must provide as input to UCBINFO. Authorized callers must serialize (such as through pinning) the device against dynamic deletions when specifying any of the following:

- DEVINFO with the UCBPTR parameter

UCBINFO Macro

- PATHINFO with the UCBPTR parameter
- PATHMAP with the UCB pointer in the MAPAREA field.
- PRFXDATA with the UCBPTR parameter
- PAVINFO with the UCBPTR parameter

See *z/OS MVS Programming: Authorized Assembler Services Guide* for information about accessing and pinning UCBs.

The caller must include the appropriate mapping macro for the UCBINFO option being used:

Option	Mapping Macro
DEVCOUNT	None
DEVINFO	IOSDDEVI mapping macro
PATHINFO	IOSDPATH mapping macro
PATHMAP	IOSDMAP mapping macro
PRFXDATA	IOSDUPI mapping macro
PAVINFO	IOSDPAVA mapping macro

See *z/OS MVS Data Areas, Vol 2 (DCCB-ITZYRETC)*.

Restrictions

None.

Input Register Information

Before issuing the UCBINFO macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0	A reason code; otherwise, used as a work register by the system
1	Used as a work register by the system
2-13	Unchanged
14	Used as a work register by the system
15	A return code

When control returns to the caller, the ARs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

Performance Implications

None.

UCBINFO DEVCOUNT

Use the UCBINFO DEVCOUNT macro to obtain a count of the UCBs for a device class.

Syntax

The standard form of the DEVCOUNT option of the UCBINFO macro is written as follows:

<i>name</i>	<i>name</i> : symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede UCBINFO.
UCBINFO	
b	One or more blanks must follow UCBINFO.

DEVCOUNT	
,COUNT= <i>count addr</i>	<i>count addr</i> : RS-type address or register (2) - (12).
,GROUP=DEVICELASS	
,DEVCLASS=ALL	Default: ALL
,DEVCLASS=CHAR	
,DEVCLASS=COMM	
,DEVCLASS=CTC	
,DEVCLASS=DASD	
,DEVCLASS=DISP	
,DEVCLASS=TAPE	
,DEVCLASS=UREC	
,GROUP=OTHER	
,DEVGROU=PAVBASE	Default: PAVBASE
,DEVGROU=PAVALIAS	
,IOTOKEN= <i>ioctoken addr</i>	<i>ioctoken addr</i> : RX-type address or register (2) - (12).
,LINKAGE=SYSTEM	Default: SYSTEM
,LINKAGE=BRANCH	
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	Default: IMPLIED_VERSION
,PLISTVER= <i>plistver</i>	<i>plistver</i> : 2
,RETCODE= <i>retcode addr</i>	<i>retcode addr</i> : RX-type address or register (2) - (12).
,RSNCODE= <i>rsncode addr</i>	<i>rsncode addr</i> : RX-type address or register (2) - (12).

Parameters

The parameters are explained as follows:

UCBINFO Macro

DEVCOUNT

Specifies that the system is to return a count of the UCBs.

,COUNT=*count addr*

Specifies the address of the fullword field that is to receive the count.

,GROUP=DEVICECLASS

GROUP specifies the grouping upon which the UCB count is based.

DEVICECLASS indicates that the UCB count is based on device classes.

,DEVICECLASS=ALL|CHAR|COMM|CTC|DASD|DISP|TAPE|UREC

Specifies the device class for which the corresponding UCBs are to be counted:

ALL Counts UCBs for all device classes

CHAR Counts UCBs for character reader device class

COMM

Counts UCBs for communications device class

CTC Counts UCBs for channel to channel device class

DASD Counts UCBs for direct access device class

DISP Counts UCBs for display device class

TAPE Counts UCBs for tape device class

UREC Counts UCBs for unit record device class

,GROUP=OTHER

GROUP specifies the grouping upon which the UCB count is based.

OTHER indicates that the UCB count is not based on device classes.

,DEVGROUP=PAVBASE

,DEVGROUP=PAVALIAS

Specifies the device group for which the corresponding UCBs are to be counted.

- **PAVBASE**, counts UCBs for Parallel Access Volume (PAV) base UCBs.

- **PAVALIAS**, counts UCBs for Parallel Access Volume (PAV) alias UCBs.

,IOCTOKEN=*ioctoken addr*

Specifies the address of a 48-character storage area that contains the MVS I/O configuration token. The caller can obtain this token by issuing the IOCINFO macro, which is described in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*. If the I/O configuration token that is current when UCBINFO is invoked does not match the token whose address is supplied here, the system issues a return code to the caller.

If you set the input IOCTOKEN (specified by *ioctoken addr*) to binary zeros, UCBINFO sets IOCTOKEN to the current I/O configuration token.

,LINKAGE=SYSTEM

,LINKAGE=BRANCH

Specifies the type of call that the system is to generate:

- **SYSTEM**: Specifies a Program Call (PC)
- **BRANCH**: Specifies a Branch entry

LINKAGE=BRANCH is intended for performance-sensitive programs.

,PLISTVER=IMPLIED_VERSION

,PLISTVER=MAX

,PLISTVER=*plistver*

Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of

the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **2**, if you use the currently available parameters.

To code, specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 2.

,RETCODE=retcode addr

Specifies the address of a fullword field into which the system copies the return code from GPR 15.

,RSNCODE=rsncode addr

Specifies the address of a fullword field into which the system copies the reason code from GPR 0.

Return and Reason Codes

When the UCBINFO DEVCOUNT macro returns control to your program, GPR 15 (or *retcode addr*, if you coded RETCODE) contains a return code, and GPR 0 (or *rsncode addr*, if you coded RSNCODE) contains a reason code.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	Meaning: The DEVCOUNT function completed successfully. Action: None.
08	01	Meaning: Program error. A caller in AR mode specified an ALET that was not valid. Action: Correct the ALET and reissue the macro.
08	02	Meaning: Program error. The system could not access the caller's parameter list. Action: Check to see if your program inadvertently overlaid the parameter list generated by the macro.
08	03	Meaning: Program error. The UCB address provided by the caller does not represent a valid UCB. Action: Correct the UCB address and reissue the macro.
08	05	Meaning: Program error. An error occurred when the system referenced the caller-supplied area specified in the IOCTOKEN parameter. This reason code is valid only for callers using the IOCTOKEN parameter. Action: Correct the IOCTOKEN parameter.

UCBINFO Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
0C	None	Meaning: Environmental error. The I/O configuration token supplied through the IOCTOKEN parameter is not current. This return code is valid only for callers using the IOCTOKEN parameter. Action: Obtain the current I/O configuration token by issuing an IOCINFO macro or by setting the input IOCTOKEN parameter in the UCBINFO macro to zero.
20	None	Meaning: System error. An unexpected error occurred. Action: Supply the return code to the appropriate IBM support personnel.

Example

To invoke UCBINFO to return a count of all DASD devices, code:

```
UCBINFO  DEVCOUNT,COUNT=CTAREA,DEVCLASS=DASD,          X
          RETCODE=INFORTCD,RSNCODE=RSNCD
          .
          .
          .
          DS  0D
CTAREA   DS  F
INFORTCD DS  F
RSNCD    DS  F
```

UCBINFO DEVCOUNT—List Form

Use the list form of the DEVCOUNT option of the UCBINFO macro together with the execute form for applications that require reentrant code. The list form of the macro defines an area of storage that the execute form uses to contain the parameters.

This macro is an alternative list form macro, and requires a different technique for using the list form as compared to the conventional list form macros. See “Alternative List Form Macros” on page 12 for further information.

The list form of the DEVCOUNT option of the UCBINFO macro is written as follows:

<i>name</i>	<i>name</i> : symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede UCBINFO.
UCBINFO	
b	One or more blanks must follow UCBINFO.

,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	Default: IMPLIED_VERSION
,PLISTVER= <i>plistver</i>	<i>plistver</i> : 2

MF=(L, <i>list addr</i>)	<i>list addr</i> . RX-type address
MF=(L, <i>list addr</i> , <i>attr</i>)	<i>attr</i> . 1- to 60-character input string
MF=(L, <i>list addr</i> ,0D)	Default: 0D

Parameters

The parameters are explained under the standard form of UCBINFO DEVCOUNT with the following exceptions:

MF=(L,*list addr*)
MF=(L,*list addr*,*attr*)
MF=(L,*list addr*,0D)

Specifies the list form of the UCBINFO DEVCOUNT macro.

list addr is the name of a storage area to contain the parameters.

attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

UCBINFO DEVCOUNT—Execute Form

Use the execute form of the DEVCOUNT option of the UCBINFO macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

The execute form of the DEVCOUNT option of the UCBINFO macro is written as follows:

<i>name</i>	<i>name</i> : symbol. Begin <i>name</i> in column 1.
	One or more blanks must precede UCBINFO.
UCBINFO	
	One or more blanks must follow UCBINFO.

DEVCOUNT

,COUNT=*count addr* *count addr*. RS-type address or register (2) - (12).

,GROUP=DEVICELASS

 ,DEVCLASS=ALL **Default:** ALL

UCBINFO Macro

,DEVCLASS=CHAR	
,DEVCLASS=COMM	
,DEVCLASS=CTC	
,DEVCLASS=DASD	
,DEVCLASS=DISP	
,DEVCLASS=TAPE	
,DEVCLASS=UREC	
,GROUP=OTHER	
,DEVGROU=PAVBAS	
,IOCTOKEN= <i>ioc</i> token <i>addr</i>	
,LINKAGE=SYSTEM	
,LINKAGE=BRANCH	
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	
,PLISTVER= <i>plist</i> ver	
,RETCODE= <i>ret</i> code <i>addr</i>	
,RSNCODE= <i>rsn</i> code <i>addr</i>	
,MF=(E, <i>list</i> <i>addr</i>)	
,MF=(E, <i>list</i> <i>addr</i> ,COMPLETE)	

Default: PAVBASE

*ioc*token *addr*: RX-type address or register (2) - (12).

Default: SYSTEM

Default: IMPLIED_VERSION

*plist*ver: 2

*ret*code *addr*: RX-type address or register (2) - (12).

*rsn*code *addr*: RX-type address or register (2) - (12).

list *addr*: RX-type address or address in register (2) - (12).

Default: COMPLETE

Parameters

The parameters are explained under the standard form of UCBINFO DEVCOUNT with the following exceptions:

,MF=(E,*list* *addr*)

,MF=(E,*list* *addr*,COMPLETE)

Specifies the execute form of the UCBINFO DEVCOUNT macro.

list *addr* specifies the area that the system uses to contain the parameters.

COMPLETE, which is the default, specifies that the macro is to check for required parameters and supply defaults for omitted optional parameters.

UCBINFO DEVINFO

Use the UCBINFO DEVINFO macro to obtain information about a device, specifically, reasons why the device is offline.

Syntax

The standard form of the DEVINFO option of the UCBINFO macro is written as follows:

<i>name</i>	<i>name</i> : symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede UCBINFO.
UCBINFO	
b	One or more blanks must follow UCBINFO.

DEVINFO

,DEVIAREA= <i>deviarea addr</i>	<i>deviarea addr</i> : RX-type address or register (2) - (12).
,DEVN= <i>devn addr</i>	<i>devn addr</i> : RS-type address or register (2) - (12).
,UCBPTR= <i>ucbptr</i>	<i>ucbptr</i> : RS-type address.
	Note: Specify either DEVN or UCBPTR, but not both.
,IOCTOKEN= <i>ioctoken addr</i>	<i>ioctoken addr</i> : RX-type address or register (2) - (12).
,LINKAGE=SYSTEM ,LINKAGE=BRANCH	Default: SYSTEM
,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX ,PLISTVER= <i>plistver</i>	Default: IMPLIED_VERSION <i>plistver</i> : 2
,RETCODE= <i>retcode addr</i>	<i>retcode addr</i> : RX-type address or register (2) - (12).
,RSNCODE= <i>rsncode addr</i>	<i>rsncode addr</i> : RX-type address or register (2) - (12).

Parameters

The parameters are explained as follows:

DEVINFO

Specifies that the system is to return information about the specified UCB.

,DEVIAREA=*deviarea addr*

Specifies the address of a required 256-byte output field into which the system is to return information about the specified UCB. This field is mapped by the mapping macro IOSDDEVI.

,DEVN=*devn addr*

Specifies the address of a halfword that contains, in binary form, the device number of the device. The DEVN and UCBPTR parameters are mutually exclusive.

,UCBPTR=*ucbptr*

Specifies that address of a fullword that contains the address of the UCB common segment. The DEVN and UCBPTR parameters are mutually exclusive.

UCBINFO Macro

,IOCTOKEN=*ioctoken addr*

Specifies the address of a 48-character storage area that contains the MVS I/O configuration token. The caller can obtain this token by issuing the IOCINFO macro, which is described in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*. If the I/O configuration token that is current when UCBINFO is invoked does not match the token whose address is supplied here, the system issues a return code to the caller.

If you set the input IOCTOKEN (specified by *ioctoken addr*) to binary zeros, UCBINFO sets IOCTOKEN to the current I/O configuration token.

,LINKAGE=SYSTEM

,LINKAGE=BRANCH

Specifies the type of call that the system is to generate:

- **SYSTEM:** Specifies a Program Call (PC)
- **BRANCH:** Specifies a Branch entry

LINKAGE=BRANCH is intended for performance-sensitive programs.

,PLISTVER=IMPLIED_VERSION

,PLISTVER=MAX

,PLISTVER=*plistver*

Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **2**, if you use the currently available parameters.

To code, specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 2

,RETCODE=*retcode addr*

Specifies the address of a fullword field into which the system copies the return code from GPR 15.

,RSNCODE=*rsncode addr*

Specifies the address of a fullword field into which the system copies the reason code from GPR 0.

Return and Reason Codes

When the UCBINFO DEVINFO macro returns control to your program, GPR 15 (or *retcode addr*, if you coded RETCODE) contains a return code, and GPR 0 (or *rsncode addr*, if you coded RSNCODE) contains a reason code.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	<p>Meaning: The DEVINFO function completed successfully.</p> <p>Action: None.</p>
04	None	<p>Meaning: Program error. No UCB exists for the device number specified in the DEVN parameter.</p> <p>Action: Correct the device number and reissue the macro.</p>
08	01	<p>Meaning: Program error. A caller in AR mode specified an ALET that was not valid.</p> <p>Action: Correct the ALET and reissue the macro.</p>
08	02	<p>Meaning: Program error. An error occurred when the system tried to access the caller's parameter list.</p> <p>Action: Ensure that you have met the environmental requirements for the macro, and reissue the macro.</p>
08	03	<p>Meaning: Program error.</p> <p>Action: Correct the UCB address and reissue the macro.</p>
08	05	<p>Meaning: Program error. An error occurred when the system referenced the caller-supplied area specified in the IOCTOKEN parameter. This reason code is valid only for callers using the IOCTOKEN parameter.</p> <p>Action: Correct the IOCTOKEN parameter.</p>
08	09	<p>Meaning: Program error. An error occurred when the system attempted to reference the area specified by the DEVIAREA parameter.</p> <p>Action: Correct the address specified on the DEVIAREA parameter and reissue the macro.</p>
0C	None	<p>Meaning: Environmental error. The I/O configuration token supplied through the IOCTOKEN parameter is not current. This return code is valid only for callers using the IOCTOKEN parameter.</p> <p>Action: Obtain the current I/O configuration token by issuing an IOCINFO macro or by setting the input IOCTOKEN parameter in the UCBINFO macro to zero.</p>
20	None	<p>Meaning: System error. An unexpected error occurred.</p> <p>Action: Supply the return code to the appropriate IBM support personnel.</p>
28	None	<p>Meaning: Program error. The device number or UCB address provided by the caller represents an alias UCB of a parallel access volume. For information about a parallel access volume, the caller must specify the base device number or base UCB.</p> <p>Action: Correct the DEVN or UCBPTR parameter and reissue the macro.</p>

UCBINFO Macro

Example

To invoke UCBINFO to return device information, code:

```
UCBINFO  DEVINFO,DEVIAREA=INFOAREA,DEVN=DEVNUM,      X
         RETCODE=INFORTCD
         .
         .
         .
         DS  0D
INFOAREA DS  CL256
INFORTCD DS  F
DEVNUM   DS  H
```

UCBINFO DEVINFO—List Form

Use the list form of the DEVINFO option of the UCBINFO macro together with the execute form for applications that require reentrant code. The list form of the macro defines an area of storage that the execute form uses to contain the parameters.

This macro is an alternative list form macro, and requires a different technique for using the list form as compared to the conventional list form macros. See “Alternative List Form Macros” on page 12 for further information.

The list form of the DEVINFO option of the UCBINFO macro is written as follows:

<i>name</i>	<i>name</i> : symbol. Begin <i>name</i> in column 1.
	One or more blanks must precede UCBINFO.
UCBINFO	
	One or more blanks must follow UCBINFO.

,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	Default: IMPLIED_VERSION
,PLISTVER= <i>plistver</i>	<i>plistver</i> : 2
MF=(L, <i>list addr</i>)	<i>list addr</i> : RX-type address
MF=(L, <i>list addr, attr</i>)	<i>attr</i> : 1- to 60-character input string
MF=(L, <i>list addr</i> ,0D)	Default: 0D

Parameters

The parameters are explained under the standard form of UCBINFO DEVINFO with the following exceptions:

MF=(L,*list addr*)
MF=(L,*list addr,attr*)

MF=(L,*list addr*,0D)

Specifies the list form of the UCBINFO DEVINFO macro.

list addr is the name of a storage area to contain the parameters.

attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

UCBINFO DEVINFO—Execute Form

Use the execute form of the DEVINFO option of the UCBINFO macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

The execute form of the DEVINFO option of the UCBINFO macro is written as follows:

<i>name</i>	<i>name</i> : symbol. Begin <i>name</i> in column 1.
␣	One or more blanks must precede UCBINFO.
UCBINFO	
␣	One or more blanks must follow UCBINFO.

DEVINFO	
,DEVIAREA= <i>deviarea addr</i>	<i>deviarea addr</i> : RX-type address or register (2) - (12).
,DEVN= <i>devn addr</i>	<i>devn addr</i> : RS-type address or register (2) - (12).
,UCBPTR= <i>ucbptr</i>	<i>ucbptr</i> : RS-type address.
	Note: Specify either DEVN or UCBPTR, but not both.
,IOCTOKEN= <i>ioctoken addr</i>	<i>ioctoken addr</i> : RX-type address or register (2) - (12).
,LINKAGE=SYSTEM	Default: SYSTEM
,LINKAGE=BRANCH	
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	Default: IMPLIED_VERSION
,PLISTVER= <i>plistver</i>	<i>plistver</i> : 2
,RETCODE= <i>retcode addr</i>	<i>retcode addr</i> : RX-type address or register (2) - (12).
,RSNCODE= <i>rsncode addr</i>	<i>rsncode addr</i> : RX-type address or register (2) - (12).
,MF=(E, <i>list addr</i>)	<i>list addr</i> : RX-type address or address in register (2) - (12).

UCBINFO Macro

,MF=(E,*list addr*,COMPLETE)

Default: COMPLETE

Parameters

The parameters are explained under the standard form of UCBINFO DEVINFO with the following exceptions:

,MF=(E,*list addr*)

,MF=(E,*list addr*,COMPLETE)

Specifies the execute form of the UCBINFO DEVINFO macro.

list addr specifies the area that the system uses to contain the parameters.

COMPLETE, which is the default, specifies that the macro is to check for required parameters and supply defaults for omitted optional parameters.

UCBINFO PATHINFO

Use the UCBINFO PATHINFO macro to obtain information about the device path and type of channel path associated with the device.

Syntax

The standard form of the PATHINFO option of the UCBINFO macro is written as follows:

<i>name</i>	<i>name</i> : symbol. Begin <i>name</i> in column 1.
␣	One or more blanks must precede UCBINFO.
UCBINFO	
␣	One or more blanks must follow UCBINFO.

PATHINFO	
,PATHAREA= <i>patharea addr</i>	<i>patharea addr</i> : RX-type address or register (2) - (12).
,DEVN= <i>devn addr</i>	<i>devn addr</i> : RS-type address or register (2) - (12).
,UCBPTR= <i>ucbptr</i>	<i>ucbptr</i> : RS-type address.
	Note: Specify either DEVN or UCBPTR, but not both.
,IOCTOKEN= <i>ioctoken addr</i>	<i>ioctoken addr</i> : RX-type address or register (2) - (12).
,LINKAGE=SYSTEM	Default: SYSTEM
,LINKAGE=BRANCH	
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	Default: IMPLIED_VERSION

<code>,PLISTVER=plistver</code>	<i>plistver</i> : 2
<code>,RETCODE=retcode addr</code>	<i>retcode addr</i> : RX-type address or register (2) - (12).
<code>,RSNCODE=rsncode addr</code>	<i>rsncode addr</i> : RX-type address or register (2) - (12).

Parameters

The parameters are explained as follows:

PATHINFO

Specifies that the system is to return information about the device path and type of channel path for the specified UCB.

,PATHAREA=patharea addr

Specifies the address of the required 256-byte output field into which the system is to return information about the device path and type of channel path for the specified UCB. This field is mapped by the mapping macro IOSDPATH.

,DEVN=devn addr

Specifies the address of a halfword that contains, in binary form, the device number of the device.

,UCBPTR=ucbptr

Specifies the address of a fullword that contains the address of the UCB common segment. The caller can obtain the address of the UCB common segment by a UCBPTR parameter on a UCBLOOK macro.

,IOCTOKEN=ioctoken addr

Specifies the address of a 48-character storage area that contains the MVS I/O configuration token. The caller can obtain this token by issuing the IOCINFO macro, which is described in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*. If the I/O configuration token that is current when UCBINFO is invoked does not match the token whose address is supplied here, the system issues a return code to the caller.

If you set the input IOCTOKEN (specified by *ioctoken addr*) to binary zeros, UCBINFO sets IOCTOKEN to the current I/O configuration token.

,LINKAGE=SYSTEM

,LINKAGE=BRANCH

Specifies the type of call that the system is to generate:

- **SYSTEM:** Specifies a Program Call (PC)
- **BRANCH:** Specifies a Branch entry

LINKAGE=BRANCH is intended for performance sensitive programs.

,PLISTVER=IMPLIED_VERSION

,PLISTVER=MAX

,PLISTVER=plistver

Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

UCBINFO Macro

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **2**, if you use the currently available parameters.

To code, specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 2

,RETCODE=*retcode addr*

Specifies the address of a fullword field into which the system copies the return code from GPR 15.

,RSNCODE=*rsncode addr*

Specifies the address of a fullword field into which the system copies the reason code from GPR 0.

Return and Reason Codes

When the UCBINFO PATHINFO macro returns control to your program, GPR 15 (or *retcode addr*, if you coded RETCODE) contains a return code, and GPR 0 (or *rsncode addr*, if you coded RSNCODE) contains a reason code.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	Meaning: The PATHINFO function completed successfully. Action: None.
04	None	Meaning: Program error. No UCB exists for the device number specified in the DEVN parameter. Action: Correct the device number and reissue the macro.
08	01	Meaning: Program error. A caller in AR mode specified an ALET that was not valid. Action: Correct the ALET and reissue the macro.
08	02	Meaning: Program error. An error occurred when the system tried to access the caller's parameter list. Action: Ensure that you have met the environmental requirements for the macro, and reissue the macro.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
08	03	<p>Meaning: Program error.</p> <p>The UCB address provided by the caller does not represent a valid UCB.</p> <p>Action: Correct the UCB address and reissue the macro.</p>
08	05	<p>Meaning: Program error. An error occurred when the system referenced the caller-supplied area specified in the IOCTOKEN parameter. This reason code is valid only for callers using the IOCTOKEN parameter.</p> <p>Action: Correct the IOCTOKEN parameter.</p>
08	08	<p>Meaning: Program error. An error occurred when the system attempted to reference the area specified by the PATHAREA parameter.</p> <p>Action: Correct the address specified on the PATHAREA parameter and reissue the macro.</p>
0C	None	<p>Meaning: Environmental error. The I/O configuration token supplied through the IOCTOKEN parameter is not current. This return code is valid only for callers using the IOCTOKEN parameter.</p> <p>Action: Obtain the current I/O configuration token by issuing an IOCINFO macro or by setting the input IOCTOKEN parameter in the UCBINFO macro to zero.</p>
18	04	<p>Meaning: System error. The subchannel is in permanent error and cannot be accessed.</p> <p>Action: Supply the return and reason codes to the appropriate IBM support personnel.</p>
18	08	<p>Meaning: Environmental error. The UCB is not connected to a subchannel.</p> <p>Action: Verify that there is a device at the device number associated with the subchannel, and reissue the macro.</p>
20	None	<p>Meaning: System error. An unexpected error occurred.</p> <p>Action: Supply the return code to the appropriate IBM support personnel.</p>

Example

To invoke UCBINFO to return device path and type of channel path information, code:

```

UCBINFO  PATHINFO,PATHAREA=INFOAREA,DEVN=DEVNUM,          X
          RETCODE=INFORTCD
          .
          .
          .

          DS  0D
INFOAREA DS  CL256
INFORTCD DS  F
DEVNUM   DS  H

```

UCBINFO PATHINFO—List Form

Use the list form of the PATHINFO option of the UCBINFO macro together with the execute form for applications that require reentrant code. The list form of the macro defines an area of storage that the execute form uses to contain the parameters.

This macro is an alternative list form macro, and requires a different technique for using the list form as compared to the conventional list form macros. See “Alternative List Form Macros” on page 12 for further information.

The list form of the PATHINFO option of the UCBINFO macro is written as follows:

<i>name</i>	<i>name</i> : symbol. Begin <i>name</i> in column 1.
	One or more blanks must precede UCBINFO.
UCBINFO	
	One or more blanks must follow UCBINFO.

,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	Default: IMPLIED_VERSION
,PLISTVER= <i>plistver</i>	<i>plistver</i> : 2
MF=(L, <i>list addr</i>)	<i>list addr</i> : RX-type address
MF=(L, <i>list addr</i> , <i>attr</i>)	<i>attr</i> : 1- to 60-character input string
MF=(L, <i>list addr</i> ,0D)	Default: 0D

Parameters

The parameters are explained under the standard form of UCBINFO PATHINFO with the following exceptions:

MF=(L,*list addr*)

MF=(L,*list addr*,*attr*)

MF=(L,*list addr*,0D)

Specifies the list form of the UCBINFO PATHINFO macro.

list addr is the name of a storage area to contain the parameters.

attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

UCBINFO PATHINFO—Execute Form

Use the execute form of the PATHINFO option of the UCBINFO macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

The execute form of the PATHINFO option of the UCBINFO macro is written as follows:

<i>name</i>	<i>name</i> : symbol. Begin <i>name</i> in column 1.
␣	One or more blanks must precede UCBINFO.
UCBINFO	
␣	One or more blanks must follow UCBINFO.

PATHINFO	
,PATHAREA= <i>patharea addr</i>	<i>patharea addr</i> : RX-type address or register (2) - (12).
,DEVN= <i>devn addr</i>	<i>devn addr</i> : RS-type address or register (2) - (12).
,UCBPTR= <i>ucbptr</i>	<i>ucbptr</i> : RS-type address.
	Note: Specify either DEVN or UCBPTR, but not both.
,IOCTOKEN= <i>ioctoken addr</i>	<i>ioctoken addr</i> : RX-type address or register (2) - (12).
,LINKAGE=SYSTEM	Default: SYSTEM
,LINKAGE=BRANCH	
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	Default: IMPLIED_VERSION
,PLISTVER= <i>plistver</i>	<i>plistver</i> : 2
,RETCODE= <i>retcode addr</i>	<i>retcode addr</i> : RX-type address or register (2) - (12).
,RSNCODE= <i>rsncode addr</i>	<i>rsncode addr</i> : RX-type address or register (2) - (12).
,MF=(E, <i>list addr</i>)	<i>list addr</i> : RX-type address or address in register (2) - (12).
,MF=(E, <i>list addr</i> ,COMPLETE)	Default: COMPLETE

Parameters

The parameters are explained under the standard form of UCBINFO PATHINFO with the following exceptions:

,MF=(E,*list addr*)

UCBINFO Macro

$$MF = (E, list\ addr, COMPLETE)$$

Specifies the execute form of the UCBINFO PATHINFO macro.

list addr specifies the area that the system uses to contain the parameters.

COMPLETE, which is the default, specifies that the macro is to check for required parameters and supply defaults for omitted optional parameters.

UCBINFO PATHMAP

Use the `UCBINFO PATHMAP` macro to obtain information about the device path.

Syntax

The standard form of the PATHMAP option of the UCBINFO macro is written as follows:

<i>name</i>	<i>name</i> : symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede UCBINFO.
UCBINFO	
b	One or more blanks must follow UCBINFO.
<hr/>	
PATHMAP	
,MAPAREA= <i>maparea addr</i>	<i>maparea addr</i> : RX-type address or register (2) - (12).
,DEVN= <i>devn addr</i>	<i>devn addr</i> : RS-type address or register (2) - (12).
,DEVN=NONE	Default: NONE
,IOCTOKEN= <i>ioctoken addr</i>	<i>ioctoken addr</i> : RX-type address or register (2) - (12).
,LINKAGE=SYSTEM	Default: SYSTEM
,LINKAGE=BRANCH	
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	Default: IMPLIED_VERSION
,PLISTVER= <i>plistver</i>	<i>plistver</i> : 2
,RETCODE= <i>retcode addr</i>	<i>retcode addr</i> : RX-type address or register (2) - (12).
,RSNCODE= <i>rsncode addr</i>	<i>rsncode addr</i> : RX-type address or register (2) - (12).

Parameters

The parameters are explained as follows:

PATHMAP

Specifies that the system is to return information about the device path for the specified UCB.

,MAPAREA=maparea addr

Specifies a required 40-byte field into which the system is to return information about the device path for the specified UCB. This field is mapped by the mapping macro IOSDMAP.

,DEVN=devn addr**,DEVN=NONE**

Specifies the address of a halfword that contains, in binary form, the device number of the device.

If the caller does not specify an address on the DEVN parameter, or specifies DEVN=NONE, the caller must place the address of the UCB common segment into the fullword field within the MAPAREA DSECT that is assigned the name MAPUCB by mapping macro IOSDMAP. See *z/OS MVS Programming: Authorized Assembler Services Guide* for information about using UCBSCAN to obtain the address of the UCB.

If the caller codes the DEVN parameter, the MAPUCB field contains hexadecimal zeros when control returns to the caller.

,IOCTOKEN=ioctoken addr

Specifies the address of a 48-character storage area that contains the MVS I/O configuration token. The caller can obtain this token by issuing the IOCINFO macro, which is described in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*. If the I/O configuration token that is current when UCBINFO is invoked does not match the token whose address is supplied here, the system issues a return code to the caller.

If you set the input IOCTOKEN (specified by *ioctoken addr*) to binary zeros, UCBINFO sets IOCTOKEN to the current I/O configuration token.

,LINKAGE=SYSTEM**,LINKAGE=BRANCH**

Specifies the type of call that the system is to generate:

- **SYSTEM:** Specifies a Program Call (PC)
- **BRANCH:** Specifies a Branch entry

LINKAGE=BRANCH is intended for performance-sensitive programs.

,PLISTVER=IMPLIED_VERSION**,PLISTVER=MAX****,PLISTVER=plistver**

Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that

UCBINFO Macro

the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **2**, if you use the currently available parameters.

To code, specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 2

,RETCODE=*retcode addr*

Specifies the address of a fullword field into which the system copies the return code from GPR 15.

,RSNCODE=*rsncode addr*

Specifies the address of a fullword field into which the system copies the reason code from GPR 0.

Return and Reason Codes

When the UCBINFO PATHMAP macro returns control to your program, GPR 15 (or *retcode addr*, if you coded RETCODE) contains a return code, and GPR 0 (or *rsncode addr*, if you coded RSNCODE) contains a reason code.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	Meaning: The PATHMAP function completed successfully. Action: None.
04	None	Meaning: Program error. No UCB exists for the device number specified in the DEVN parameter. Action: Correct the device number and reissue the macro.
08	01	Meaning: Program error. A caller in AR mode specified an ALET that was not valid. Action: Correct the ALET and reissue the macro.
08	02	Meaning: Program error. An error occurred when the system tried to access the caller's parameter list. Action: Ensure that you have met the environmental requirements for the macro, and reissue the macro.
08	03	Meaning: Program error. The UCB address provided by the caller does not represent a valid UCB. Action: Correct the UCB address and reissue the macro.
08	05	Meaning: Program error. An error occurred when the system referenced the caller-supplied area specified in the IOCTOKEN parameter. This reason code is valid only for callers using the IOCTOKEN parameter. Action: Correct the IOCTOKEN parameter.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
08	06	<p>Meaning: Program error. An error occurred when the system attempted to reference the area specified by the MAPAREA parameter.</p> <p>Action: Correct the address specified for MAPAREA and reissue the macro.</p>
0C	None	<p>Meaning: Environmental error. The I/O configuration token supplied through the IOCTOKEN parameter is not current. This return code is valid only for callers using the IOCTOKEN parameter.</p> <p>Action: Obtain the current I/O configuration token by issuing an IOCINFO macro or by setting the input IOCTOKEN parameter in the UCBINFO macro to zero.</p>
10	04	<p>Meaning: System error. The subchannel is in permanent error and cannot be accessed.</p> <p>Action: Supply the return and reason code to the appropriate IBM support personnel.</p>
10	08	<p>Meaning: Environmental error. The UCB is not connected to a subchannel.</p> <p>Action: Correct the UCB address supplied, and reissue the macro.</p>
20	None	<p>Meaning: System error. An unexpected error occurred.</p> <p>Action: Supply the return code to the appropriate IBM support personnel.</p>

Example

To invoke UCBINFO to return device path information, code:

```

        UCBINFO  PATHMAP,MAPAREA=INFOAREA,DEVN=DEVNUM,          X
        RETCODE=INFORTCD
        .
        .
        .
        DS      0D
INFOAREA DS      CL256
INFORTCD DS      F
DEVNUM   DS      H

```

UCBINFO PATHMAP—List Form

Use the list form of the PATHMAP option of the UCBINFO macro together with the execute form for applications that require reentrant code. The list form of the macro defines an area of storage that the execute form uses to contain the parameters.

This macro is an alternative list form macro, and requires a different technique for using the list form as compared to the conventional list form macros. See “Alternative List Form Macros” on page 12 for further information.

The list form of the PATHMAP option of the UCBINFO macro is written as follows:

UCBINFO Macro

<i>name</i>	<i>name</i> : symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede UCBINFO.
UCBINFO	
b	One or more blanks must follow UCBINFO.

,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	Default: IMPLIED_VERSION
,PLISTVER= <i>plistver</i>	<i>plistver</i> : 2
MF=(L, <i>list addr</i>)	<i>list addr</i> : RX-type address
MF=(L, <i>list addr</i> , <i>attr</i>)	<i>attr</i> : 1- to 60-character input string
MF=(L, <i>list addr</i> ,0D)	Default: 0D

Parameters

The parameters are explained under the standard form of UCBINFO PATHMAP with the following exceptions:

MF=(L,*list addr*)

MF=(L,*list addr*,*attr*)

MF=(L,*list addr*,0D)

Specifies the list form of the UCBINFO PATHMAP macro.

list addr is the name of a storage area to contain the parameters.

attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

UCBINFO PATHMAP—Execute Form

Use the execute form of the PATHMAP option of the UCBINFO macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

The execute form of the PATHMAP option of the UCBINFO macro is written as follows:

<i>name</i>	<i>name</i> : symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede UCBINFO.

UCBINFO

b One or more blanks must follow UCBINFO.

PATHMAP

,MAPAREA= <i>maparea addr</i>	<i>maparea addr</i> : RX-type address or register (2) - (12).
,DEVN= <i>devn addr</i>	<i>devn addr</i> : RX-type address or register (2) - (12).
,DEVN=NONE	Default: NONE
,IOCTOKEN= <i>ioctoken addr</i>	<i>ioctoken addr</i> : RX-type address or register (2) - (12).
,LINKAGE=SYSTEM	Default: SYSTEM
,LINKAGE=BRANCH	
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	Default: IMPLIED_VERSION
,PLISTVER= <i>plistver</i>	<i>plistver</i> : 2
,RETCODE= <i>retcode addr</i>	<i>retcode addr</i> : RX-type address or register (2) - (12).
,RSNCODE= <i>rsncode addr</i>	<i>rsncode addr</i> : RX-type address or register (2) - (12).
,MF=(E, <i>list addr</i>)	<i>list addr</i> : RX-type address or address in register (2) - (12).
,MF=(E, <i>list addr</i> ,COMPLETE)	Default: COMPLETE

Parameters

The parameters are explained under the standard form of the UCBINFO PATHMAP macro with the following exceptions:

,MF=(E,*list addr*)

,MF=(E,*list addr*,COMPLETE)

Specifies the execute form of the UCBINFO PATHMAP macro.

list addr specifies the area that the system uses to contain the parameters.

COMPLETE, which is the default, specifies that the macro is to check for required parameters and supply defaults for omitted optional parameters.

UCBINFO PAVINFO

Use the UCBINFO PAVINFO macro to obtain selected information applicable to each exposure (base and alias) of a Parallel Access Volume (PAV).

Syntax

The standard form of the PAVINFO option of the UCBINFO macro is written as follows:

UCBINFO Macro

<i>name</i>	<i>name</i> : symbol. Begin <i>name</i> in column 1.
<i>b</i>	One or more blanks must precede UCBINFO.
UCBINFO	
<i>b</i>	One or more blanks must follow UCBINFO.

PAVINFO

PAVINFOSUM=NO PAVINFOSUM=YES	Default: NO
,PAVAREA= <i>pavarea addr</i>	<i>pavarea addr</i> : RX-type address or register (2) - (12).
,PAVLEN= <i>pavarea length addr</i>	<i>pavarea lenth addr</i> : RX-type address or register (2) - (12).
,SCHINFO=NO ,SCHINFO=YES	Default: NO
,DEVN= <i>devn addr</i>	<i>devn addr</i> : RS-type address or register (2) - (12).
,UCBPTR= <i>ucbptr</i>	<i>ucbptr</i> : RX-type address. Note: Specify either DEVN or UCBPTR, but not both.
,IOCTOKEN= <i>ioctoken addr</i>	<i>ioctoken addr</i> : RX-type address or register (2) - (12).
,LINKAGE=SYSTEM ,LINKAGE=BRANCH	Default: SYSTEM
,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX ,PLISTVER= <i>plistver</i>	Default: IMPLIED_VERSION <i>plistver</i> : 2
,RETCODE= <i>retcode addr</i>	<i>retcode addr</i> : RX-type address or register (2) - (12).
,RSNCODE= <i>rsncode addr</i>	<i>rsncode addr</i> : RX-type address or register (2) - (12).

Parameters

The parameters are explained as follows:

PAVINFO

Obtain selected information that applies to each exposure of a Parallel Access Volume (PAV) device. The data returned by this function is an array. Depending on the input device, the following is returned:

- When the input device is a PAV-base, the first array entry represents the base and each subsequent array entry represents each of the bound PAV-alias devices associated with the base. Note that if the base has no bound PAV-aliases, then only the first array entry is filled in.
- When the input is a non-PAV DASD device, only the first array entry is filled in.
- When the input device is a PAV-alias or a non-DASD, a non-zero return code is returned.

PAVINFOSUM=NO**PAVINFOSUM=YES**

Specifies whether to retrieve only a sum of channel measurement data and model dependent subchannel data for the base device and all of its aliases.

Note: The model dependent subchannel data is only retrieved if SCHINFO=YES.

NO Do not just retrieve a total of channel measurement data and model dependent subchannel data for the base device and all of its aliases. This option causes each element of the PAVA array to contain information for the base device and each of its aliases.

YES Retrieve only a sum of channel measurement data and model dependent subchannel data for the base device and all of its aliases. This option causes the first element of the PAVA array to contain information on the base device, however, the PAVACMB and PAVASMDB fields will contain totals for the base and all of its aliases.

,PAVAREA=*pavarea addr*

Specifies the address of a required output field into which the system will return information about the alias UCBs for the specified base device number or base UCB address. This field is mapped by the mapping macro IOSDPAVA.

,PAVLEN=*pavarea lengthaddr*

Specifies the address or a register containing the length of the area specified by the PAVAREA parameter.

,SCHINFO=NO**,SCHINFO=YES**

Specifies whether to retrieve model-dependent subchannel data for the device.

NO Do not retrieve model-dependent subchannel data for the device.

YES Retrieve model-dependent subchannel data for the device.

,DEVN=*devn addr*

Specifies the address of a halfword that contains the base device number in binary form. The DEVN and UCBPTR parameters are mutually exclusive.

,UCBPTR=*ucbptr*

Specifies the address of a fullword that contains the address of the UCB common segment. The DEVN and UCBPTR parameters are mutually exclusive.

,IOCTOKEN=*ioctoken addr*

Specifies the address of a 48-character storage area that contains the MVS I/O configuration token. The caller can obtain this token by issuing the IOCINFO macro, which is described in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*. If the I/O configuration token that is current when UCBINFO is invoked does not match the token whose address is supplied here, the system issues a return code to the caller.

UCBINFO Macro

If you set the input IOCTOKEN (specified by *ioctoken addr*) to binary zeros, UCBINFO sets IOCTOKEN to the current I/O configuration token.

,LINKAGE=SYSTEM

,LINKAGE=BRANCH

Specifies the type of call that the system is to generate:

- **SYSTEM:** Specifies a program call (PC)
- **BRANCH:** Specifies a branch entry

LINKAGE=BRANCH is intended for performance sensitive programs.

,PLISTVER=IMPLIED_VERSION

,PLISTVER=MAX

,PLISTVER=*plistver*

Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **2**, if you use the currently available parameters.

To code, specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value in the range of 1 - 3.

,RETCODE=*retcode addr*

Specifies the address of a fullword field into which the system copies the return code from GPR 15.

,RSNCODE=*rsncode addr*

Specifies the address of a fullword field into which the system copies the reason code from GPR 0.

Return and Reason Codes

When the UCBINFO PAVINFO macro returns control to your program, GPR 15 (or *retcode addr*, if you coded RETCODE) contains a return code, and GPR 0 (or *rsncode addr*, if you coded RSNCODE) contains a reason code.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	Meaning: The PAVINFO function completed successfully. Action: None.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
04	None	<p>Meaning: Program error. No UCB exists for the device number specified in the DEVN parameter.</p> <p>Action: Correct the device number and reissue the macro.</p>
08	01	<p>Meaning: Program error. A caller in AR mode specified an ALET that was not valid.</p> <p>Action: Correct the ALET and reissue the macro.</p>
08	02	<p>Meaning: Program error. An error occurred when the system tried to access the caller's parameter list.</p> <p>Action: Ensure that you have met the environmental requirements for the macro, and reissue the macro.</p>
08	03	<p>Meaning: Program error.</p> <p>The UCB address provided by the caller does not represent a valid UCB.</p> <p>Action: Correct the UCB address and reissue the macro.</p>
08	05	<p>Meaning: Program error. An error occurred when the system referenced the caller-supplied area specified in the IOCTOKEN parameter. This reason code is valid only for callers using the IOCTOKEN parameter.</p> <p>Action: Correct the IOCTOKEN parameter and reissue the macro.</p>
08	0A	<p>Meaning: Program error. An error occurred when the system attempted to reference the area specified by the PAVAREA parameter.</p> <p>Action: Correct the address specified on the PAVAREA parameter and reissue the macro.</p>
0C	None	<p>Meaning: Environmental error. The I/O configuration token supplied through the IOCTOKEN parameter is not current. This return code is valid only for callers using the IOCTOKEN parameter.</p> <p>Action: Obtain the current I/O configuration token by issuing an IOCINFO macro or by setting the input IOCTOKEN parameter in the UCBINFO macro to zero.</p>
1C	01	<p>Meaning: Program error. The device number or UCB address provided by the caller specifies a device that is not a DASD or is a PAV alias device.</p> <p>Action: Correct the DEVN or UCBPTR parameter and reissue the macro.</p>
1C	02	<p>Meaning: Program error. The work area specified with the PAVAREA parameter is not large enough to contain the minimum amount of data. No data is returned.</p> <p>Action: Increase the size of the specified work area and reissue the macro.</p>

UCBINFO Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
1C	03	Meaning: Program error. The work area specified with the PAVAREA parameter is not large enough to contain an array element for each alias device. Action: Increase the size of the specified work area and reissue the macro.
20	None	Meaning: System error. An unexpected error occurred. Action: Supply the return code to the appropriate IBM support personnel.
28	None	Meaning: Program error. The device number or UCB address provided by the caller represents an alias UCB of a parallel access volume. The caller must specify the base device number or base UCB. Action: Correct the DEVN or UCBPTR parameter and reissue the macro.

Example

To invoke UCBINFO to return information about alias UCBs for a base device number, code:

```
        UCBINFO PAVINFO,DEVN=DEVNUM,PAVAREA=INFOAREA,PAVLEN=AREALEN,  X
        RETCODE=INFORTCD
        .
        .
        .
        DS 0D
DEVNUM DS H
INFOAREA DS CL256
AREALEN DS F
INFORTCD DS F
```

To invoke UCBINFO to return information about alias UCBs for a base UCB, code:

```
        UCBINFO PAVINFO,UCBPTR=UCBP,PAVAREA=INFOAREA,PAVLEN=AREALEN,  X
        RETCODE=INFORTCD
        .
        .
        .
        DS 0D
UCBP   DS A
INFOAREA DS CL256
AREALEN DS F
INFORTCD DS F
```

UCBINFO PAVINFO—List Form

Use the list form of the PAVINFO option of the UCBINFO macro together with the execute form for applications that require reentrant code. The list form of the macro defines an area of storage that the execute form uses to contain the parameters.

The list form of the PAVINFO option of the UCBINFO macro is written as follows:

<i>name</i>	<i>name</i> : symbol. Begin <i>name</i> in column 1.
b UCBINFO	One or more blanks must precede UCBINFO.
b	One or more blanks must follow UCBINFO.

,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	Default: IMPLIED_VERSION
,PLISTVER= <i>plistver</i>	<i>plistver</i> : 2
MF=(L, <i>list addr</i>)	<i>list addr</i> : RX-type address
MF=(L, <i>list addr</i> , <i>attr</i>)	<i>attr</i> : 1- to 60-character input string
MF=(L, <i>list addr</i> ,0D)	Default: 0D

Parameters

The parameters are explained under the standard form of UCBINFO PAVINFO with the following exceptions:

MF=(L,*list addr*)

MF=(L,*list addr*,*attr*)

MF=(L,*list addr*,0D)

Specifies the list form of the UCBINFO PAVINFO macro.

list addr is the name of a storage area to contain the parameters.

attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of X'0D', which forces the parameter list to a doubleword boundary.

UCBINFO PAVINFO—Execute Form

Use the execute form of the PAVINFO option of the UCBINFO macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

The execute form of the PAVINFO option of the UCBINFO macro is written as follows:

<i>name</i>	<i>name</i> : symbol. Begin <i>name</i> in column 1.
b UCBINFO	One or more blanks must precede UCBINFO.

UCBINFO Macro

b One or more blanks must follow UCBINFO.

PAVINFO

PAVINFORM=NO PAVINFORM=YES	Default: NO
,PAVAREA= <i>pavarea addr</i>	<i>pavarea addr</i> : RX-type address or register (2) - (12).
,PAVLEN= <i>pavarea length addr</i>	<i>pavarea length addr</i> : RX-type address or register (2) - (12).
,SCHINFO=NO ,SCHINFO=YES	Default: NO
,DEVN= <i>devn addr</i>	<i>devn addr</i> : RX-type address or register (2) - (12).
,UCBPTR= <i>ucbptr</i>	<i>ucbptr</i> : RX-type address. Note: Specify either DEVN or UCBPTR, but not both.
,IOCTOKEN= <i>ioctoken addr</i>	<i>ioctoken addr</i> : RX-type address or register (2) - (12).
,LINKAGE=SYSTEM ,LINKAGE=BRANCH	Default: SYSTEM
,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX ,PLISTVER= <i>plistver</i>	Default: IMPLIED_VERSION <i>plistver</i> : 2
,RETCODE= <i>retcode addr</i>	<i>retcode addr</i> : RX-type address or register (2) - (12).
,RSNCODE= <i>rsncode addr</i>	<i>rsncode addr</i> : RX-type address or register (2) - (12).
,MF=(E, <i>list addr</i>) ,MF=(E, <i>list addr</i> ,COMPLETE)	<i>list addr</i> : RX-type address or address in register (2) - (12). Default: COMPLETE

Parameters

The parameters are explained under the standard form of UCBINFO PAVINFO with the following exceptions:

,MF=(E,*list addr*)

,MF=(E,*list addr*,COMPLETE)

Specifies the execute form of the UCBINFO PAVINFO macro.

list addr specifies the area that the system uses to contain the parameters.

COMPLETE, which is the default, specifies that the macro is to check for required parameters and supply defaults for omitted optional parameters.

UCBINFO PRFXDATA

Use the UCBINFO PRFXDATA macro to obtain a copy of the UCB prefix extension segment.

Syntax

The standard form of the PRFXDATA option of the UCBINFO macro is written as follows:

<i>name</i>	<i>name</i> : symbol. Begin <i>name</i> in column 1.
␣	One or more blanks must precede UCBINFO.
UCBINFO	
␣	One or more blanks must follow UCBINFO.

PRFXDATA	
,DEVN= <i>devn addr</i>	<i>devn addr</i> : RS-type address or register (2) - (12).
,UCBPTR= <i>ucbptr addr</i>	<i>ucbptr addr</i> : RS-type address or register (2) - (12).
	Note: Specify either DEVN or UCBPTR, but not both.
,UCBPAREA= <i>ucbparea addr</i>	<i>ucbparea addr</i> : RX-type address or register (2) - (12).
,IOCTOKEN= <i>ioctoken addr</i>	<i>ioctoken addr</i> : RX-type address or register (2) - (12).
,LINKAGE=SYSTEM	Default: SYSTEM
,LINKAGE=BRANCH	
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	Default: IMPLIED_VERSION
,PLISTVER= <i>plistver</i>	<i>plistver</i> : 2
,RETCODE= <i>retcode addr</i>	<i>retcode addr</i> : RX-type address or register (2) - (12).
,RSNCODE= <i>rsncode addr</i>	<i>rsncode addr</i> : RX-type address or register (2) - (12).

Parameters

The parameters are explained as follows:

PRFXDATA

Specifies that the system is to obtain information from the UCB prefix extension segment.

UCBINFO Macro

,DEVN=*devn addr*

Specifies the address of a halfword that contains, in binary form, the device number of the device.

,UCBPTR=*ucbptr addr*

Specifies the address of a fullword that contains the address of the UCB common segment. The caller can obtain the address of the UCB common segment by a UCBPTR parameter on a UCBLOOK macro.

,UCBPAREA=*ucbparea addr*

Specifies the address of a 48-character storage area into which the system copies the UCB prefix extension segment. The IOSDUPI mapping macro maps the area.

,IOCTOKEN=*ioctoken addr*

Specifies the address of a 48-character storage area that contains the MVS I/O configuration token. The caller can obtain this token by issuing the IOCINFO macro, which is described in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*. If the I/O configuration token that is current when UCBINFO is invoked does not match the token whose address is supplied here, the system issues a return code to the caller.

If you set the input IOCTOKEN (specified by *ioctoken addr*) to binary zeros, UCBINFO sets IOCTOKEN to the current I/O configuration token.

,LINKAGE=SYSTEM

,LINKAGE=BRANCH

Specifies the type of call that the system is to generate:

- **SYSTEM:** Specifies a Program Call (PC)
- **BRANCH:** Specifies a Branch entry

LINKAGE=BRANCH is intended for performance-sensitive programs.

,PLISTVER=IMPLIED_VERSION

,PLISTVER=MAX

,PLISTVER=*plistver*

Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **2**, if you use the currently available parameters.

To code, specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 2

,RETCODE=*retcode addr*

Specifies the address of a fullword field into which the system copies the return code from GPR 15.

,RSNCODE=*rsncode addr*

Specifies the address of a fullword field into which the system copies the reason code from GPR 0.

Return and Reason Codes

When the UCBINFO PRFXDATA macro returns control to your program, GPR 15 (or *retcode addr*, if you coded RETCODE) contains a return code, and GPR 0 (or *rsncode addr*, if you coded RSNCODE) contains a reason code.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	Meaning: The PRFXDATA function completed successfully. Action: None.
04	None	Meaning: Program error. No UCB exists for the device number specified in the DEVN parameter. Action: Correct the device number and reissue the macro.
08	01	Meaning: Program error. A caller in AR mode specified an ALET that was not valid. Action: Correct the ALET and reissue the macro.
08	02	Meaning: Program error. An error occurred when the system tried to access the caller's parameter list. Action: Ensure that you have met the environmental requirements for the macro, and reissue the macro.
08	03	Meaning: Program error. The UCB address provided by the caller does not represent a valid UCB. Action: Correct the UCB address and reissue the macro.
08	05	Meaning: Program error. An error occurred when the system referenced the caller-supplied area specified in the IOCTOKEN parameter. This reason code is valid only for callers using the IOCTOKEN parameter. Action: Correct the IOCTOKEN parameter.
0C	None	Meaning: Environmental error. The I/O configuration token supplied through the IOCTOKEN parameter is not current. This return code is valid only for callers using the IOCTOKEN parameter. Action: Obtain the current I/O configuration token by issuing an IOCINFO macro or by setting the input IOCTOKEN parameter in the UCBINFO macro to zero.

UCBINFO Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
20	None	Meaning: System error. An unexpected error occurred. Action: Supply the return code to the appropriate IBM support personnel.

Example

To invoke UCBINFO to obtain a copy of the UCB prefix extension segment, code:

```
UCBINFO  PRFXDATA,DEVN=DEVNUM,UCBPAREA=UAREA,          X
          RETCODE=INFORTCD
          .
          .
          .
          DS  0D
DEVNUM    DS  H
UAREA     DS  CL48
INFORTCD  DS  F
```

UCBINFO PRFXDATA—List Form

Use the list form of the PRFXDATA option of the UCBINFO macro together with the execute form for applications that require reentrant code. The list form of the macro defines an area of storage that the execute form uses to contain the parameters.

This macro is an alternative list form macro, and requires a different technique for using the list form as compared to the conventional list form macros. See “Alternative List Form Macros” on page 12 for further information.

The list form of the PRFXDATA option of the UCBINFO macro is written as follows:

<i>name</i>	<i>name</i> : symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede UCBINFO.
UCBINFO	
b	One or more blanks must follow UCBINFO.

,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	Default: IMPLIED_VERSION
,PLISTVER= <i>plistver</i>	<i>plistver</i> : 2
MF=(L, <i>list addr</i>)	<i>list addr</i> : RX-type address
MF=(L, <i>list addr</i> , <i>attr</i>)	<i>attr</i> : 1- to 60-character input string
MF=(L, <i>list addr</i> ,0D)	Default: 0D

Parameters

The parameters are explained under the standard form of UCBINFO PRFXDATA with the following exceptions:

MF=(L,*list addr*)

MF=(L,*list addr*,*attr*)

MF=(L,*list addr*,0D)

Specifies the list form of the UCBINFO PRFXDATA macro.

list addr is the name of a storage area to contain the parameters.

attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

UCBINFO PRFXDATA—Execute Form

Use the execute form of the PRFXDATA option of the UCBINFO macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

The execute form of the PRFXDATA option of the UCBINFO macro is written as follows:

<i>name</i>	<i>name</i> : symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede UCBINFO.
UCBINFO	
b	One or more blanks must follow UCBINFO.

PRFXDATA

,DEVN= <i>devn addr</i>	<i>devn addr</i> : RS-type address or register (2) - (12).
,UCBPTR= <i>ucbptr addr</i>	<i>ucbptr addr</i> : RS-type address or register (2) - (12).
	Note: Specify either DEVN or UCBPTR, but not both.
,UCBPAREA= <i>ucbparea addr</i>	<i>ucbparea addr</i> : RX-type address or register (2) - (12).
,IOCTOKEN= <i>ioctoken addr</i>	<i>ioctoken addr</i> : RX-type address or register (2) - (12).
,LINKAGE=SYSTEM	Default: SYSTEM
,LINKAGE=BRANCH	

UCBINFO Macro

<code>,PLISTVER=IMPLIED_VERSION</code>	
<code>,PLISTVER=MAX</code>	Default: IMPLIED_VERSION
<code>,PLISTVER=<i>plistver</i></code>	<i>plistver</i> : 2
<code>,RETCODE=<i>retcode addr</i></code>	<i>retcode addr</i> : RX-type address or register (2) - (12).
<code>,RSNCODE=<i>rsncode addr</i></code>	<i>rsncode addr</i> : RX-type address or register (2) - (12).
<code>,MF=(E,<i>list addr</i>)</code>	<i>list addr</i> : RX-type address or address in register (2) - (12).
<code>,MF=(E,<i>list addr</i>,COMPLETE)</code>	Default: COMPLETE

Parameters

The parameters are explained under the standard form of UCBINFO PRFXDATA with the following exceptions:

,MF=(E,*list addr*)

,MF=(E,*list addr*,COMPLETE)

Specifies the execute form of the UCBINFO PRFXDATA macro.

list addr specifies the area that the system uses to contain the parameters.

COMPLETE, which is the default, specifies that the macro is to check for required parameters and supply defaults for omitted optional parameters.

UCBLOOK — Obtain Addresses of UCB Segments

Description

The UCBLOOK macro obtains the address of the following for a given device number or volume serial number:

- The UCB common segment
- The UCB common extension segment
- The UCB prefix extension segment

The input device number may be specified as binary or EBCDIC, and may be a 3-digit or 4-digit number.

You can use UCBLOOK to locate any UCB segment, including a segment for a dynamic UCB. The caller must pin the UCB by means of the PIN parameter unless one of the following is true:

- The caller is running in an environment where dynamic I/O configuration changes cannot occur.
- The caller can otherwise guarantee that the UCB will not be deleted.

After the system returns the address of the UCB segment, and the caller is done processing the UCB, the caller must unpin the UCB. The caller can unpin the UCB by using the UCBPIN macro with the UNPIN parameter.

Note: The caller can optionally restrict the search to UCBs that are static and installation-static, have 3 digit device numbers, or are below 16 megabytes.

Environment

The requirements for the caller are:

Minimum authorization:	Supervisor state or PKM keys 0-7 For LINKAGE=BRANCH, all of the following: <ul style="list-style-type: none">• Supervisor state with PSW key 0• 31-bit addressing mode• Primary ASC mode• Parameter list and any data areas it points to must be in fixed storage or, if the caller is disabled, in disabled reference (DREF) storage
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	No requirement
Control parameters:	Must be in the primary address space or be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL)

Programming Requirements

If in AR mode, specify SYSSTATE ASCENV=AR before invoking the macro.

UCBLOOK Macro

Restrictions

None.

Input Register Information

Before issuing the UCBLOOK macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output Register Information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register	Contents
0	Reason code if GPR 15 contains a return code of 08; otherwise, used as a work register by the system
1	Used as a work register by the system
2-13	Unchanged
14	Used as a work register by the system
15	Return code

When control returns to the caller, the access registers (ARs) contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

Performance Implications

None.

Syntax

The standard form of the UCBLOOK macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
␣	One or more blanks must precede UCBLOOK
UCBLOOK	
␣	One or more blanks must follow UCBLOOK

DEVN= <i>devn addr</i>	<i>devn addr</i> : RS-type address or register (2) - (12).
DEVNCHAR= <i>devnchar addr</i>	<i>devnchar addr</i> : RS-type address or register (2) - (12).
VOLSER= <i>volser addr</i>	<i>volser addr</i> : RS-type address or register (2) - (12).
,UCBPTR= <i>ucbptr addr</i>	<i>ucbptr addr</i> : RS-type address or register (2) - (12).
,UCBCXPTR= <i>ucbcxptr addr</i>	<i>ucbcxptr addr</i> : RS-type address or register (2) - (12).
,UCBPXPTR= <i>ucbpxptr addr</i>	<i>ucbpxptr addr</i> : RS-type address or register (2) - (12).

,UCBPAREA= <i>ucbparea addr</i> ,UCBPAREA=NONE	<i>ucbparea addr</i> : RX-type address or register (2) - (12). Default: NONE
,LOC=BELOW ,LOC=ANY	Default: BELOW
,PIN ,NOPIN	Note: TEXT and PTOKEN are required with PIN and are not valid with NOPIN.
,TEXT= <i>text addr</i>	<i>text addr</i> : RX-type address
,PTOKEN= <i>ptoken addr</i>	<i>ptoken addr</i> : RS-type address or register (2) - (12).
,LASTING	Note: Optional with PIN; not valid with NOPIN.
,UNBOUND_ALIAS=NO ,UNBOUND_ALIAS=YES	Default: NO
,DEVCLASS=DASD ,DEVCLASS=DASDTAPE ,DEVCLASS=TAPE	Note: DEVCLASS is valid only with VOLSER= <i>volser addr</i> Default: DASDTAPE
,DYNAMIC=NO ,DYNAMIC=YES	Default: NO
,RANGE=3DIGIT ,RANGE=ALL	Default: 3DIGIT
,IOCTOKEN= <i>ioctoken addr</i> ,IOCTOKEN=NONE	<i>ioctoken addr</i> : RX-type address or register (2) - (12). Default: NONE
,LINKAGE=SYSTEM ,LINKAGE=BRANCH	Default: SYSTEM
,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX ,PLISTVER= <i>plistver</i>	Default: IMPLIED_VERSION <i>plistver</i> : 2
,RETCODE= <i>retcode addr</i>	<i>retcode addr</i> : RX-type address or register (2) - (12).
,RSNCODE= <i>rsncode addr</i>	<i>rsncode addr</i> : RX-type address or register (2) - (12).

Parameters

The parameters are explained as follows:

DEVN=*devn addr*

DEVNCHAR=*devnchar addr*

VOLSER=*volser addr*

Specifies the address of an input field that identifies the device whose UCB address is to be obtained.

UCBLOOK Macro

DEVN specifies the address of a halfword that contains, in binary form, the device number of the device whose UCB address is to be obtained.

DEVNCHAR specifies the address of a 4-character field that contains, in EBCDIC, the device number of the device whose UCB address is to be obtained.

Note: A 3-digit device number can be represented as either 'ddd' or '0ddd' where ddd is the device number.

VOLSER specifies the address of a 6-character field that contains, in EBCDIC, the volume serial number of the device whose UCB address is to be obtained.

,UCBPTR=ucbptr addr

Specifies the address of a fullword field in which the address of the UCB common segment associated with the requested device (DEVN, DEVNCHAR, or VOLSER) will be returned.

,UCBCXPTR=ucbcxptr addr

Specifies the address of a fullword field in which the system returns the address of the UCB common extension segment associated with the specified device (DEVN, DEVNCHAR, or VOLSER). Use the IEFUCBOB mapping macro to map the UCB common extension segment.

,UCBPXPTR=ucbpxptr addr

Specifies the address of a fullword field in which the system returns the address of the UCB prefix extension segment associated with the specified device (DEVN, DEVNCHAR, or VOLSER). Use the IOSDUPFX mapping macro to map the UCB prefix extension segment.

,UCBPAREA=ucbparea addr

,UCBPAREA=NONE

Specifies the address of a 48-character storage area that will receive a copy of the UCB prefix extension segment. The copy of the UCB prefix extension segment is mapped by the IOSDUP1 mapping macro.

,LOC=BELOW

,LOC=ANY

Specifies whether the search should be restricted to below 16 megabyte UCB (LOC=BELOW) or should also include above 16 megabyte UCBs (LOC=ANY).

,PIN

,NOPIN

Specifies whether the UCB is to be pinned to make it ineligible for deletion through dynamic I/O configuration changes. Pinning the UCB ensures that it cannot be deleted while the look-up process is taking place. The PIN parameter specifies that the UCB should be pinned, and NOPIN specifies that it should not. Programs that pin a UCB are also responsible for unpinning it once the UCB is no longer subject to processing. Use the UCBPIN macro with the UNPIN option to unpin the UCB.

,TEXT=text addr

Specifies the address of a 58-character input field containing text that documents the reason for the PIN request. If the pin request remains outstanding during a request for a configuration change that would delete this UCB, the text specified by the TEXT parameter will be displayed in a message identifying the reason for a configuration change failure.

,PTOKEN=*ptoken addr*

Specifies the address of an 8-character area that is to receive the pin token for the UCB. The caller must use the pin token when unpinning the UCB through the UCBPIN service.

,LASTING

Specifies that the UCB will not be unpinned automatically by the system at the time of termination of the task or address space with which the pin is associated.

When you code LASTING, the system cannot dynamically delete the UCB until your program issues UCBPIN with the UNPIN parameter.

,UNBOUND_ALIAS=NO**,UNBOUND_ALIAS=YES**

Specifies whether the scan should include unbound alias UCBs.

YES Include unbound alias UCBs

NO Do not include unbound alias UCBs

Note: The UNBOUND_ALIAS function is intended for IOS use only.

,DEVCLASS=DASD**,DEVCLASS=DASDTAPE****,DEVCLASS=TAPE**

Specifies the device class that is to be searched for the VOLSER look-up.

DASD Searches UCBs for direct access device class

DASDTAPE Searches UCBs for DASD and tape classes

TAPE Searches UCBs for tape class

,DYNAMIC=NO**,DYNAMIC=YES**

Specifies if static or dynamic UCBs are to be looked at:

NO Only static and installation-static UCBs

YES Static, installation-static, and dynamic UCBs

,RANGE=3DIGIT**,RANGE=ALL**

Specifies whether the look-up should be restricted to UCBs with 3-digit device numbers (3DIGIT) or should also include UCBs with 4-digit device numbers (ALL).

,IOCTOKEN=*ioctoken addr***,IOCTOKEN=NONE**

Specifies the address of a 48-character area that contains the MVS I/O configuration token that the caller supplies to UCBLOOK. Obtain this token by issuing the IOCINFO macro, which is described in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*. If the I/O configuration token that is current when UCBLOOK is invoked does not match the token whose address is supplied as input by *ioctoken addr*, the caller will be notified through a return code.

If the input IOCTOKEN (specified by *ioctoken addr*) is set to binary zeros, UCBLOOK will set IOCTOKEN to the current I/O configuration token.

,LINKAGE=SYSTEM**,LINKAGE=BRANCH**

Specifies the type of call that should be generated:

- **SYSTEM:** Specifies a Program Call (PC)
- **BRANCH:** Specifies a branch entry

UCBLOOK Macro

LINKAGE=BRANCH is intended for performance-sensitive programs.

,PLISTVER=IMPLIED_VERSION

,PLISTVER=MAX

,PLISTVER=*plistver*

Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **2**, if you use the currently available parameters.

To code, specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 2

,RETCODE=*retcode addr*

Specifies the fullword location where the system is to store the return code. The return code is also in GPR 15.

,RSNCODE=*rsncode addr*

Specifies the fullword location where the system is to store the reason code. The reason code is also in GPR 0.

ABEND Codes

None.

Return and Reason Codes

When control returns from UCBLOOK, GPR 15 (and *retcode addr*, if you coded RETCODE) contains a return code and, for return code X'08', GPR 0 (and *rsncode addr*, if you coded RSNCODE) contains a reason code.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	Meaning: Completed successfully. Action: None.
04	None	Meaning: Program error. No UCB exists for the device number or VOLSER specified as input. The contents of UCBPTR remain unchanged. Action: Correct the DEVN, DEVNCHAR, or VOLSER parameter. Also, make sure the parameter list was not overlaid.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
08	01	<p>Meaning: Program error. An ALET in the parameter list is not valid.</p> <p>Action: Make sure the parameter list was not overlaid.</p>
08	02	<p>Meaning: Program error. An error occurred in accessing the caller's parameter list.</p> <p>Action: Make sure the parameter list was not overlaid.</p>
08	04	<p>Meaning: Program error. An error occurred in referencing the caller-supplied area for the UCB prefix extension segment. This reason code is valid only for callers using the UCBPAREA parameter.</p> <p>Action: Correct the UCBPAREA parameter.</p>
08	05	<p>Meaning: Program error. An error occurred in referencing the caller-supplied area for the IOCTOKEN. This reason code is valid only for callers using the IOCTOKEN keyword.</p> <p>Action: Correct the IOCTOKEN parameter.</p>
08	0A	<p>Meaning: Program error. An error occurred in referencing the caller-supplied area for the pin reason text. This reason code is valid only for callers using the TEXT parameter.</p> <p>Action: Correct the TEXT parameter.</p>
0C	None	<p>Meaning: Program error. The UCB definition for the device specified in DEVN, DEVNCHAR, or VOLSER is not longer consistent with the UCB definition represented by the input I/O configuration token, or a DDR has occurred. This return code is valid only for callers using the IOCTOKEN parameter.</p> <p>Action: Change the IOCTOKEN parameter or change the program so that the token is correct.</p>
20	None	<p>Meaning: System error. An unexpected error occurred.</p> <p>Action: Supply the return code to the appropriate IBM support personnel.</p>

UCBLOOK—List Form

Use the list form of the UCBLOOK macro together with the execute form for applications that require reentrant code. The list form of the macro defines an area of storage that the execute form uses for storing the parameters.

Syntax

This macro is an alternative list form macro, and requires a different technique for using the list form as compared to the conventional list form macros. See “Alternative List Form Macros” on page 12 for further information.

The list form of the UCBLOOK macro is written as follows:

UCBLOOK Macro

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
<i>b</i>	One or more blanks must precede UCBLOOK
UCBLOOK	
<i>b</i>	One or more blanks must follow UCBLOOK

<i>,PLISTVER=IMPLIED_VERSION</i>	
<i>,PLISTVER=MAX</i>	Default: IMPLIED_VERSION
<i>,PLISTVER=plistver</i>	<i>plistver</i> : 2
MF=(L, <i>list addr</i>)	<i>list addr</i> : symbol.
MF=(L, <i>list addr,attr</i>)	<i>attr</i> : 1- to 60-character input string.
MF=(L, <i>list addr,0D</i>)	Default: 0D

Parameters

The parameters are explained under the standard form of the UCBLOOK macro with the following exceptions:

MF=(L,*list addr*)

MF=(L,*list addr,attr*)

MF=(L,*list addr,0D*)

Specifies the list form of the UCBLOOK macro.

The *list addr* parameter specifies the address of the storage area for the parameter list.

attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

UCBLOOK—Execute Form

Use the execute form of the UCBLOOK macro together with the list form for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

Syntax

The execute form of the UCBLOOK macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
-------------	--

One or more blanks must precede UCBLOOK

UCBLOOK

One or more blanks must follow UCBLOOK

DEVN= <i>devn addr</i>	<i>devn addr</i> : RS-type address or register (2) - (12).
DEVNCHAR= <i>devnchar addr</i>	<i>devnchar addr</i> : RS-type address or register (2) - (12).
VOLSER= <i>volser addr</i>	<i>volser addr</i> : RS-type address or register (2) - (12).
,UCBPTR= <i>ucbptr addr</i>	<i>ucbptr addr</i> : RS-type address or register (2) - (12).
,UCBCXPTR= <i>ucbcxptr addr</i>	<i>ucbcxptr addr</i> : RS-type address or register (2) - (12).
,UCBPXPTR= <i>ucbpxptr addr</i>	<i>ucbpxptr addr</i> : RS-type address or register (2) - (12).
,UCBPAREA= <i>ucbparea addr</i>	<i>ucbparea addr</i> : RX-type address or register (2) - (12).
,UCBPAREA=NONE	Default: NONE
,LOC=BELOW	Default: BELOW
,LOC=ANY	
,PIN	
,NOPIN	Note: TEXT and PTOKEN are required with PIN and are not valid with NOPIN.
,TEXT= <i>text addr</i>	<i>text addr</i> : RX-type address
,PTOKEN= <i>ptoken addr</i>	<i>ptoken addr</i> : RS-type address or register (2) - (12).
,LASTING	Note: Optional with PIN; not valid with NOPIN.
,UNBOUND_ALIAS=NO	Default: NO
,UNBOUND_ALIAS=YES	
,DEVCLASS=DASD	Note: DEVCLASS is valid only with VOLSER= <i>volser addr</i>
,DEVCLASS=DASDTAPE	Default: DASDTAPE
,DEVCLASS=TAPE	
,DYNAMIC=NO	Default: NO
,DYNAMIC=YES	
,RANGE=3DIGIT	Default: 3DIGIT
,RANGE=ALL	
,IOCTOKEN= <i>ioctoken addr</i>	<i>ioctoken addr</i> : RX-type address or register (2) - (12).
,IOCTOKEN=NONE	Default: NONE
,LINKAGE=SYSTEM	Default: SYSTEM
,LINKAGE=BRANCH	
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	Default: IMPLIED_VERSION

UCBLOOK Macro

<code>,PLISTVER=plistver</code>	<i>plistver</i> : 2
<code>,RETCODE=retcode addr</code>	<i>retcode addr</i> : RX-type address or register (2) - (12).
<code>,RSNCODE=rsncode addr</code>	<i>rsncode addr</i> : RX-type address or register (2) - (12).
<code>,MF=(E,list addr)</code>	<i>list addr</i> : RX-type address or register (2) - (12).
<code>,MF=(E,list addr,COMPLETE)</code>	Default: COMPLETE

Parameters

The parameters are explained under the standard form of the UCBLOOK macro with the following exceptions:

,MF=(E,list addr)

,MF=(E,list addr,COMPLETE)

Specifies the execute form of the UCBLOOK macro.

The *list addr* parameter specifies the address of the storage area for the parameter list. COMPLETE specifies that the system is to check for required parameters and supply defaults for optional parameters that were not specified.

UCBPIN — Pinning or Unpinning a UCB

Description

Pinning a UCB ensures that the UCB cannot be deleted while a program is in the process of looking at a UCB. Programs that pin a UCB are also responsible for unpinning it once the UCB is no longer subject to processing.

Authorized programs that obtain UCB addresses, either through UCB services or other means, can use the UCBPIN macro to pin and unpin UCBs. Pinning and unpinning should be done any time a UCB is used, unless one of the following is true:

- The caller is running in an environment where dynamic configuration changes cannot occur.
- The caller can otherwise guarantee that the UCB will not be deleted. (The device is allocated.)

Movepin allows users to move pin information from one PAV UCB to another.

Note: The movepin function is intended for IOS use only.

Environment

The requirements for the caller are:

Minimum authorization:	Supervisor state or PKM keys 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN or PASN~=HASN~=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	No requirement
Control parameters:	Must be in the primary address space or be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL)

Programming Requirements

If the program is in AR mode, issue the SYSSTATE ASCENV=AR macro before UCBPIN. SYSSTATE ASCENV=AR tells the system to generate code appropriate for AR mode.

Restrictions

None.

Register Information

After the caller issues the macro, the system might use some registers as work registers or might change the contents of some registers. When the system returns control to the caller, the contents of these registers are not the same as they were before the macro was issued. Therefore, if the caller depends on these registers containing the same value before and after issuing the macro, the caller must save these registers before issuing the macro and restore them after the system returns control.

UCBPIN Macro

When control returns to the caller, the general purpose registers (GPRs) contain:

Register	Contents
0	Reason code if GPR 15 contains a return code of 08; otherwise, used as a work register by the system
1	Used as a work register by the system
2-13	Unchanged
14	Used as a work register by the system
15	Return code

When control returns to the caller, the access registers (ARs) contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

Performance Implications

None.

Syntax

The standard form of the UCBPIN macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
␣	One or more blanks must precede UCBPIN
UCBPIN	
␣	One or more blanks must follow UCBPIN

PIN	
UNPIN	Note: See the table following this diagram for valid parameter combinations.
,PTOKEN= <i>ptoken addr</i>	<i>ptoken addr</i> : RS-type address or register (2) - (12).
,UCBPTR= <i>ucbptr addr</i>	<i>ucbptr addr</i> : RS-type address or register (2) - (12).
,TEXT= <i>text addr</i>	<i>text addr</i> : RX-type address
,LASTING	
,IOCTOKEN= <i>ioctoken addr</i>	<i>ioctoken addr</i> : RX-type address or register (2) - (12).
,RETCODE= <i>retcode addr</i>	<i>retcode addr</i> : RX-type address or register (2) - (12).
,RSNCODE= <i>rsncode addr</i>	<i>rsncode addr</i> : RX-type address or register (2) - (12).
,LINKAGE=SYSTEM	Default: SYSTEM
,LINKAGE=BRANCH	

MOVEPIN

,TO=*xto* *xto*: RS-type address or register (2) - (12).

,FROM=*xfrom* *xfrom*: RS-type address or register (2) - (12).

,RETCODE=*retcode addr* *retcode addr*: RX-type address or register (2) - (12).

,RSNCODE=*rsncode addr* *rsncode addr*: RX-type address or register (2) - (12).

,LINKAGE=SYSTEM **Default:** SYSTEM

,LINKAGE=BRANCH

The following table shows how other parameters may be used with PIN and UNPIN.

Parameters	PIN	UNPIN
PTOKEN	required	required
UCBPTR	required	not valid
TEXT	required	not valid
LASTING	optional	not valid
IOCTOKEN	optional	not valid
LINKAGE	optional	optional
RETCODE	optional	optional
RSNCODE	optional	optional

Parameters	PIN	UNPIN	MOVEPIN
PTOKEN	required	required	not valid
UCBPTR	required	not valid	not valid
TEXT	required	not valid	not valid
LASTING	optional	not valid	not valid
IOCTOKEN	optional	not valid	not valid
LINKAGE	optional	optional	optional
RETCODE	optional	optional	optional
RSNCODE	optional	optional	optional
TO	not valid	not valid	required
FROM	not valid	not valid	required

Parameters

The parameters are explained as follows:

PIN

UNPIN

Specifies whether the UCB is to be pinned or unpinned.

UCBPIN Macro

,PTOKEN=*ptoken addr*

Specifies the address of an 8-character field used to contain the pin token. For PIN requests, PTOKEN specifies an output field that receives the pin token for the UCB that is to be pinned. For UNPIN requests, PTOKEN specifies an input field that contains the pin token for the UCB that is to be unpinned; this token must match the one that was returned on the corresponding PIN request. UCBPIN will reset PTOKEN to binary zeros if the UNPIN function is successful.

,UCBPTR=*ucbptr addr*

Specifies the address of a pointer containing the address of the UCB common segment for the UCB that is to be pinned.

,TEXT=*text addr*

Specifies the address of a 58-character input field containing text that documents the reason for the PIN request. If the pin request remains outstanding during a request for a configuration change that would delete this UCB, the text specified by the TEXT parameter will be displayed in a message identifying the reason for a configuration change failure.

,LASTING

Specifies that the UCB will not be unpinned automatically by the system at the time of termination of the task or address space with which the pin is associated.

When you code LASTING, the system cannot dynamically delete the UCB until your program issues UCBPIN with the UNPIN parameter.

,IOCTOKEN=*ioctoken addr*

Specifies the address of a 48-character area that contains the MVS I/O configuration token that you supply to UCBPIN. You can obtain this token by issuing the IOCINFO macro, which is described in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*. If the I/O configuration token that is current when UCBPIN is invoked does not match the token specified as input by *ioctoken addr*, the caller will be notified through a return code.

If the input IOCTOKEN (specified by *ioctoken addr*) is set to binary zeros, UCBPIN will set IOCTOKEN to the current I/O configuration token.

MOVEPIN

Specifies whether to move pin-related information from one Parallel Access Volume (PAV) to another. The FROM and TO UCBs are checked to insure that one is an active PAV-base and the other is a PAV-alias bound to that base.

Note: The MOVEPIN function is intended for IOS use only.

TO=*to*

Specifies the address of the UCB common segment of the device that the pin information is being moved to.

FROM=*from*

Specifies the address of the UCB common segment of the device that the pin information is being moved from.

,RETCODE=*retcode addr*

Specifies the fullword location where the system is to store the return code. The return code is also in GPR 15.

,RSNCODE=*rsncode addr*

Specifies the fullword location where the system is to store the reason code. The reason code is also in GPR 0 if the return code is X'08'.

,LINKAGE=SYSTEM

,LINKAGE=BRANCH

Specifies the type of call that the system is to generate:

- **SYSTEM:** Specifies a Program Call (PC)
- **BRANCH:** Specifies a Branch entry

LINKAGE=BRANCH is intended for performance-sensitive programs.

Return and Reason Codes

When control returns from UCBPIN, GPR 15 (and *retcode addr*, if you coded RETCODE) contains one of the following return codes:

Table 38. Return Codes for the UCBPIN Macro

Hexadecimal Return Code	Meaning																
00	Meaning: UCBPIN completed successfully. For PIN requests, the UCB has been pinned and the pin token has been returned in PTOKEN. For UNPIN requests, the UCB has been unpinned and PTOKEN has been reset to binary zeros.																
08	<p>Meaning: There is an error in the caller's parameters, as explained by the hexadecimal reason code that accompanies this return code. The reason code is in GPR 0 (and in <i>rsncode addr</i>, if you coded RSNCODE).</p> <table> <tr> <th>Reason Code</th><th>Meaning</th></tr> <tr> <td>01</td><td>An ALET in the parameter list is not valid; the caller might have inadvertently written over an area in the parameter list.</td></tr> <tr> <td>02</td><td>An error occurred in accessing the caller's parameter list.</td></tr> <tr> <td>03</td><td>The UCB address provided by the caller does not represent a valid UCB.</td></tr> <tr> <td>04</td><td>The PTOKEN supplied as input on an UNPIN request does not represent a valid pin token.</td></tr> <tr> <td>05</td><td>An error occurred in referencing the user-supplied work area for the IOCTOKEN. This reason code is valid only for callers using the IOCTOKEN keyword.</td></tr> <tr> <td>0A</td><td>An error occurred in referencing the user-supplied work area for the pin reason text. This reason code is valid only for callers using the TEXT keyword.</td></tr> <tr> <td>0B</td><td>MOVEPIN was requested, but the FROM and TO devices did not represent an active PAV-base and a bound PAV-alias on that base. The request is rejected. (This reason code is valid only for callers using the MOVEPIN function.)</td></tr> </table>	Reason Code	Meaning	01	An ALET in the parameter list is not valid; the caller might have inadvertently written over an area in the parameter list.	02	An error occurred in accessing the caller's parameter list.	03	The UCB address provided by the caller does not represent a valid UCB.	04	The PTOKEN supplied as input on an UNPIN request does not represent a valid pin token.	05	An error occurred in referencing the user-supplied work area for the IOCTOKEN. This reason code is valid only for callers using the IOCTOKEN keyword.	0A	An error occurred in referencing the user-supplied work area for the pin reason text. This reason code is valid only for callers using the TEXT keyword.	0B	MOVEPIN was requested, but the FROM and TO devices did not represent an active PAV-base and a bound PAV-alias on that base. The request is rejected. (This reason code is valid only for callers using the MOVEPIN function.)
Reason Code	Meaning																
01	An ALET in the parameter list is not valid; the caller might have inadvertently written over an area in the parameter list.																
02	An error occurred in accessing the caller's parameter list.																
03	The UCB address provided by the caller does not represent a valid UCB.																
04	The PTOKEN supplied as input on an UNPIN request does not represent a valid pin token.																
05	An error occurred in referencing the user-supplied work area for the IOCTOKEN. This reason code is valid only for callers using the IOCTOKEN keyword.																
0A	An error occurred in referencing the user-supplied work area for the pin reason text. This reason code is valid only for callers using the TEXT keyword.																
0B	MOVEPIN was requested, but the FROM and TO devices did not represent an active PAV-base and a bound PAV-alias on that base. The request is rejected. (This reason code is valid only for callers using the MOVEPIN function.)																
0C	Meaning: The UCB definition is not consistent with the input configuration token, or a DDR has occurred. This return code is valid only for callers using the IOCTOKEN keyword.																
20	Meaning: An unexpected error occurred.																

UCBPIN—List Form

Use the list form of the UCBPIN macro together with the execute form for applications that require reentrant code. The list form of the macro defines an area of storage that the execute form uses for storing the parameters.

UCBPIN Macro

Syntax

This macro is an alternative list form macro, and requires a different technique for using the list form as compared to the conventional list form macros. See “Alternative List Form Macros” on page 12 for further information.

The list form of the UCBPIN macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
	One or more blanks must precede UCBPIN
UCBPIN	
	One or more blanks must follow UCBPIN

MF=(L, <i>list addr</i>)	<i>list addr</i> : Symbol.
MF=(L, <i>list addr</i> , <i>attr</i>)	<i>attr</i> : 1- to 60-character input string.
MF=(L, <i>list addr</i> ,0D)	Default: 0D

Parameters

The parameters are explained as follows:

MF=(L,*list addr*)

MF=(L,*list addr*,*attr*)

MF=(L,*list addr*,0D)

Specifies the list form of the UCBPIN macro.

The *list addr* parameter specifies the address of the storage area for the parameter list.

attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

UCBPIN—Execute Form

Use the execute form of the UCBPIN macro together with the list form for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

Syntax

The execute form of the UCBPIN macro is written as follows:

Note: See the table following this diagram for valid parameter combinations.

,MF=(E,*list addr*,COMPLETE) **Default:** COMPLETE

UCBPIN Macro

Parameters	PIN	UNPIN	MOVEPIN
TEXT	required	not valid	not valid
LASTING	optional	not valid	not valid
IOCTOKEN	optional	not valid	not valid
LINKAGE	optional	optional	optional
RETCODE	optional	optional	optional
RSNCODE	optional	optional	optional
MF	required	required	required
TO	not valid	not valid	required
FROM	not valid	not valid	required

Parameters

The parameters are explained under the standard form of the UCBPIN macro with the following exceptions:

,MF=(E,*list addr*)

,MF=(E,*list addr*,COMPLETE)

Specifies the execute form of the UCBPIN macro.

The *list addr* parameter specifies the address of the storage area for the parameter list. COMPLETE specifies that the system is to check for required parameters and supply defaults for optional parameters that were not specified.

UCBSCAN — Scan UCBs

Description

Use the UCBSCAN macro to scan unit control blocks (UCBs) and return a copy of a UCB or a UCB address on each invocation.

Two types of scans are available with UCBSCAN: A scan of all UCBs, and a scan of all UCBs within a particular device class. For each type of scan, the caller may optionally:

- Restrict the scan to UCBs defined as static or installation-static.
- Restrict the scan to UCBs with 3-digit device numbers.
- Restrict the scan to addresses of below 16 megabyte UCBs
- Request nonbase exposures of a multiple-exposure device, supported on systems prior to MVS/ESA SP 5.2.
- Request alias UCBs for a parallel access volume.
- Specify the device number with which the scan should begin.

UCBSCAN presents the UCBs in ascending device number order. UCBSCAN provides two options as follows:

COPY On each invocation, UCBSCAN returns a copy of requested UCB segments and data in caller-supplied areas.

Address

On each invocation, UCBSCAN returns the address of the UCB, the address of requested UCB segments, and, optionally, a copy of the UCB prefix extension segment. The caller can specify whether the scan includes above 16 megabyte UCBs or only below 16 megabyte UCBs. The caller must pin and unpin the UCB unless one of the following is true:

- The caller is running in an environment where dynamic configuration changes cannot occur
- The caller can otherwise guarantee that the UCB will not be deleted.

See *z/OS MVS Programming: Authorized Assembler Services Guide* for information on pinning UCBs.

LINKAGE=BRANCH is intended for performance-sensitive programs.

Environment

The requirements for the caller are:

Minimum authorization:	For the COPY parameter: Problem state with any PSW key
	For the ADDRESS parameter: Supervisor state or PKM allowing key 0-7
	For the LINKAGE=BRANCH parameter, all of the following: <ul style="list-style-type: none">• Supervisor state with key 0• 31-bit addressing mode• Primary ASC mode• Parameter list and any data areas it points to must be in fixed storage or, if the caller is disabled, in disabled reference (DREF) storage.
Dispatchable unit mode:	Task or SRB

UCBSCAN Macro

Cross memory mode:	Any PASN, any HASN, any SASN.
AMODE:	24- or 31-bit.
ASC mode:	Primary or access register (AR).
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	The caller may hold locks, but is not required to hold any.
Control parameters:	Must be in the primary address space or, for AR-mode callers, must be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming Requirements

If in AR mode, issue SYSSTATE ASCENV=AR before issuing UCBSCAN.

Restrictions

None.

Input Register Information

Before issuing the UCBSCAN macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output Register Information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register	Contents
0	Reason code if GPR 15 contains a return code of 04 or 08; otherwise, used as a work register by the system
1	Used as a work register by the system
2-13	Unchanged
14	Used as a work register by the system
15	Return code

When control returns to the caller, the access registers (ARs) contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

Performance Implications

None.

Parameters

The parameters are explained as follows:

COPY

Specifies that a copy of the UCB is to be obtained. See *z/OS HCD Planning* for a list of the MVS services that accept a UCB copy.

Note: When you issue UCBSCAN to obtain a UCB copy, the UCBID field in the copy is set to x'CC'.

,WORKAREA=workarea addr

Specifies the address of a 100-character work area used by the UCBSCAN service. The caller must initialize this work area to binary zeros before starting a UCB scan. On subsequent invocations of UCBSCAN within the same scan, the caller must leave the contents of this work area unchanged.

,UCBAREA=ucbarea addr

Specifies the address of a 48-character storage area that will receive a copy of the UCB common segment and the UCB device-dependent segment. See *z/OS HCD Planning* for a list of the MVS services that accept a UCB copy.

The caller does not need to initialize this area. Use the IEFUCBOB mapping macro to map the area. The contents of certain fields in the copy are:

- The UCBEXTP field contains either:
 - The address of the CMXTAREA, if CMXTAREA is below 16 MB
 - 0, if CMXTAREA is above 16 MB or if the CMXTAREA parameter is not specified
- The UCBNXUCB field is 0, because this field is not valid in the UCB copy.
- Address fields in the copy might not contain valid addresses, so do not use these addresses to reference the data areas they point to.

,CMXTAREA=cmxtarea addr

,CMXTAREA=NONE

Specifies the address of a 32-character storage area that will receive a copy of the UCB common extension segment. See *z/OS HCD Planning* for a list of the MVS services that accept a UCB copy and require this segment as part of a UCB copy.

Use the UCBCMEXT DSECT in the IEFUCBOB mapping macro to map the area. If the CMXTAREA area is below 16 MB, the UCBEXTP field in the UCBAREA area contains the address of the CMXTAREA area. If the CMXTAREA area is above 16 MB, the caller must explicitly supply the address of the CMXTAREA area because the UCBEXTP field will contain 0.

The UCBEXT field contains 0 because this field is not valid in the UCB copy.

The UCBCLEXT field contains the address of the DCEAREA if the UCB has a device class extension and the caller specified the DCEAREA parameter. Otherwise, the field contains 0.

,UCBPAREA=ucbparea addr

,UCBPAREA=NONE

Specifies the address of a 48-character storage area that will receive a copy of the UCB prefix extension segment. The area can be mapped by the IOSDUPI mapping macro.

,DCEAREA=dcearea addr

,DCEAREA=NONE

Specifies the address of a storage area that will receive a copy of the UCB device class extension segment. See *z/OS HCD Planning* for a list of the MVS services that accept a UCB copy and require this segment as part of a UCB copy.

Note: If DCEAREA=NONE is coded, then DCELEN=0 must be coded. If DCEAREA=NONE is defaulted, then DCELEN does not have to be coded.

UCBSCAN Macro

,DCELEN=*length addr*

Specifies the address of a 2-byte field that contains the length of the area specified by DCEAREA. The length specified must be 1 through 256 bytes. DCELEN is required with DCEAREA.

,VOLSER=*volser addr*

,VOLSER=NONE

Specifies the address of a 6-character field that indicates, in EBCDIC, the volume serial number of the device for which a UCB copy is to be obtained.

,DEVNCHAR=*devnchar addr*

Specifies the address of a 4-character field that is to receive the EBCDIC device number associated with the UCB copy.

,DEVN=*devn addr*

,DEVN=0

Specifies (**,DEVN=***devn addr*) an input halfword that contains, in binary form, the device number with which the scan is to begin. The default, **,DEVN=0**, starts the scan with the first UCB.

,DYNAMIC=NO

,DYNAMIC=YES

Specifies whether the scan should be restricted to static and installation-static UCBs (**,DYNAMIC=NO**) or should also include dynamic UCBs (**,DYNAMIC=YES**).

,RANGE=3DIGIT

,RANGE=ALL

Specifies whether the scan should be restricted to UCBs with 3-digit device numbers (**,RANGE=3DIGIT**) or should also include UCBs with 4-digit device numbers (**,RANGE=ALL**).

,NONBASE=NO

,NONBASE=YES

Specifies whether the scan should include nonbase exposures for a multiple-exposure device, supported on systems prior to MVS/ESA SP 5.2. **,NONBASE=NO** specifies only the base exposure, and **,NONBASE=YES** specifies all exposures.

Specifies whether the scan should include bound alias UCBs for a parallel access volume. **,NONBASE=NO** specifies that bound alias UCBs will not be included. **,NONBASE=YES** specifies that bound alias UCBs will be included.

,UNBOUND_ALIAS=NO

,UNBOUND_ALIAS=YES

,UNBOUND_ALIAS=ONLY

Specifies whether the scan should include unbound alias UCBs.

YES Include unbound alias UCBs

NO Do not include unbound alias UCBs

ONLY Include only unbound alias UCBs

Note: The **,UNBOUND_ALIAS** function is intended for IOS use only.

,DEVCLASS=ALL

,DEVCLASS=CHAR

,DEVCLASS=COMM

,DEVCLASS=CTC

,DEVCLASS=DASD

,DEVCLASS=DISP

,DEVCLASS=TAPE

,DEVCLASS=UREC

Specifies the device class that is to be scanned:

ALL	Scans UCBs for all device classes
CHAR	Scans UCBs for character reader device class
COMM	Scans UCBs for communications device class
CTC	Scans UCBs for channel to channel device class
DASD	Scans UCBs for direct access device class
DISP	Scans UCBs for display device class
TAPE	Scans UCBs for tape device class
UREC	Scans UCBs for unit record device class

,DEVCID=devcid addr

Specifies the address of an 8-bit input field that contains the hexadecimal device class ID of the device class to be scanned.

If you specify DEVCID, only UCBs of the particular device class specified will be presented, and the DEVCLASS parameter is ignored.

,IOCTOKEN=ioctoken addr**,IOCTOKEN=NONE**

Specifies the address of a 48-character storage area that contains the MVS I/O configuration token. The caller can obtain this token by issuing the IOCINFO macro, which is described in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*. If the I/O configuration token that is current when UCBSCAN is invoked does not match the token whose address is supplied as input by *ioctoken addr*, the caller will be notified through a return code.

If the input IOCTOKEN (specified by *ioctoken addr*) is set to binary zeros, UCBSCAN will set IOCTOKEN to the current I/O configuration token at the start of the scan.

,LINKAGE=SYSTEM**,LINKAGE=BRANCH**

Specifies the type of call that should be generated:

- **SYSTEM:** Specifies a Program Call (PC)
- **BRANCH:** Specifies a Branch entry

LINKAGE=BRANCH is intended for performance-sensitive programs.

,PLISTVER=IMPLIED_VERSION**,PLISTVER=MAX****,PLISTVER=plistver**

Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

UCBSCAN Macro

- 1, if you use the currently available parameters.

To code, specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1

,RETCODE=*retcode addr*

Specifies the fullword location where the system is to store the return code. The return code is also in GPR 15.

,RSNCODE=*rsncode addr*

Specifies the fullword location where the system is to store the reason code. The reason code is also in GPR 0.

Return and Reason Codes

When control returns from UCBSCAN, GPR 15 (and *retcode addr*, if you coded RETCODE) contains a return code and, for some return codes, GPR 0 (or *rsncode addr*, if you coded RSNCODE) contains a reason code.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	Meaning: UCBSCAN completed successfully. Action: None.
04	01	Meaning: UCBSCAN processing ended. All UCBs that met the search criteria have been presented to the caller. The contents of UCBAREA are unchanged, and WORKAREA has been reset to binary zeros. Action: None.
08	01	Meaning: Program error. A caller in AR mode specified an ALET that was not valid. Action: Correct the ALET and reissue the macro. Possibly the caller wrote over an area in the parameter list; look for this error.
08	02	Meaning: Program error. An error occurred when the system tried to access the caller's parameter list. Action: Ensure that you have met the environmental requirements for the macro, and reissue the macro.
08	03	Meaning: Program error. An error occurred in referencing the caller-supplied area for the UCB copy; the area was specified in the UCBAREA parameter. Action: Correct the UCBAREA parameter.
08	04	Meaning: Program error. An error occurred in referencing the caller-supplied area for the UCB prefix extension segment data. This reason code is valid only for callers using the UCBPAREA parameter. Action: Correct the UCBPAREA parameter.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
08	05	<p>Meaning: Program error. An error occurred when the system referenced the caller-supplied area specified in the IOCTOKEN parameter. This reason code is valid only for callers using the IOCTOKEN parameter.</p> <p>Action: Correct the IOCTOKEN parameter.</p>
08	08	<p>Meaning: Program error. An error occurred in referencing the caller-supplied work area specified in the WORKAREA parameter.</p> <p>Action: Correct the WORKAREA parameter.</p>
08	09	<p>Meaning: Program error. An error occurred in referencing the caller-supplied CMXTAREA area. This reason code is valid only for callers using the CMXTAREA parameter.</p> <p>Action: Correct the CMXTAREA parameter.</p>
08	0B	<p>Meaning: Program error. An error occurred in referencing the caller-supplied DCEAREA area. This reason code is valid only for callers using the DCEAREA parameter.</p> <p>Action: Correct the DCEAREA parameter.</p>
08	0C	<p>Meaning: Program error. The caller specified a volume serial number that is not valid. (Note that binary zeros are not considered valid.) This reason code is valid only for callers using the VOLSER parameter.</p> <p>Action: Correct the VOLSER parameter.</p>
08	0D	<p>Meaning: Program error. For the DCEAREA token, the caller specified a length that is negative, is zero, or exceeds 256 bytes. This reason code is valid only for callers using the DCELEN parameter.</p> <p>Action: Correct the DCELEN parameter.</p>
0C	None	<p>Meaning: Environmental error. The I/O configuration has changed, so that the I/O configuration token supplied through the IOCTOKEN parameter is not current. This return code is valid only for callers using the IOCTOKEN parameter.</p> <p>Action: Obtain the current I/O configuration token by issuing an IOCINFO macro or by setting the input IOCTOKEN parameter in the UCBINFO macro to zero. Start the scan from the beginning.</p>
20	None	<p>Meaning: System error. An unexpected error occurred.</p> <p>Action: Supply the return code to the appropriate IBM support personnel.</p>

UCBSCAN COPY

Syntax

The standard form of the COPY function of the UCBSCAN macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
␣	One or more blanks must precede UCBSCAN.
UCBSCAN	
␣	One or more blanks must follow UCBSCAN.
COPY	Default: COPY
,WORKAREA= <i>workarea addr</i>	<i>workarea addr</i> : RX-type address or register (2) - (12).
,UCBAREA= <i>ucbarea addr</i>	<i>ucbarea addr</i> : RX-type address or register (2) - (12).
,CMXTAREA= <i>cmxtarea addr</i> ,CMXTAREA=NONE	<i>cmxtarea addr</i> : RX-type address or register (2) - (12). Default: NONE
,UCBPAREA= <i>ucbparea addr</i> ,UCBPAREA=NONE	<i>ucbparea addr</i> : RX-type address or register (2) - (12). Default: NONE
,DCEAREA= <i>dcearea addr</i> ,DCEAREA=NONE	<i>dcearea addr</i> : RX-type address or register (2) - (12). Default: NONE
,DCELEN= <i>length addr</i>	<i>length addr</i> : RS-type address or register (2) - (12). Note: DCELEN is valid only with DCEAREA and is required with DCEAREA.
,VOLSER= <i>volser addr</i> ,VOLSER=NONE	<i>volser addr</i> : RS-type address or register (2) - (12). Default: NONE
,DEVNCHAR= <i>devnchar</i>	<i>addr devnchar addr</i> : RS-type address or register (2) - (12).
,DEVN= <i>devn addr</i> ,DEVN=0	<i>devn addr</i> : RS-type address or register (2) - (12). Default: 0
,DYNAMIC=NO ,DYNAMIC=YES	Default: NO
,RANGE=3DIGIT ,RANGE=ALL	Default: 3DIGIT
,NONBASE=NO ,NONBASE=YES	Default: NO
,UNBOUND_ALIAS=NO	Default: NO

,UNOUND_ALIAS=YES	
,DEVCLASS=ALL	Default: ALL
,DEVCLASS=CHAR	
,DEVCLASS=COMM	
,DEVCLASS=CTC	
,DEVCLASS=DASD	
,DEVCLASS=DISP	
,DEVCLASS=TAPE	
,DEVCLASS=UREC	
,DEVCID= <i>devcid addr</i>	<i>devcid addr</i> : RS-type address
,DEVCID=0	Default: 0
,IOCTOKEN= <i>ioctoken addr</i>	<i>ioctoken addr</i> : RX-type address or register (2) - (12).
,IOCTOKEN=NONE	Default: NONE
,LINKAGE=SYSTEM	Default: LINKAGE=SYSTEM
,LINKAGE=BRANCH	
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	Default: IMPLIED_VERSION
,PLISTVER= <i>plistver</i>	<i>plistver</i> : 1
,RETCODE= <i>retcode addr</i>	<i>retcode addr</i> : RX-type address or register (2) - (12).
,RSNCODE= <i>rsncode addr</i>	<i>rsncode addr</i> : RX-type address or register (2) - (12).

UCBSCAN COPY—List Form

Use the list form of the UCBSCAN macro together with the execute form for applications that require reentrant code. The list form of the macro defines an area of storage that the execute form uses for storing the parameters.

Syntax

This macro is an alternative list form macro, and requires a different technique for using the list form as compared to the conventional list form macros. See “Alternative List Form Macros” on page 12 for further information.

The list form of the COPY function of the UCBSCAN macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
␣	One or more blanks must precede UCBSCAN.
UCBSCAN	
␣	One or more blanks must follow UCBSCAN.

UCBSCAN Macro

,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	Default: IMPLIED_VERSION
,PLISTVER= <i>plistver</i>	<i>plistver</i> : 1
MF=(L, <i>list addr</i>)	<i>list addr</i> : Symbol.
MF=(L, <i>list addr,attr</i>)	<i>attr</i> : 1- to 60-character input string.
MF=(L, <i>list addr</i> ,0D)	Default: 0D

Parameters

The parameters are explained under that standard form of the UCBSCAN macro with the following exceptions:

$$MF = (L, list\ addr)$$
$$\mathbf{MF}=(\mathbf{L}, list\ addr, attr)$$
$$\mathbf{MF} = (\mathbf{L}, list\ addr, \mathbf{0D})$$

Specifies the list form of the UCBSCAN macro.

The *list addr* parameter specifies the address of the storage area for the parameter list.

attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

UCBSCAN COPY—Execute Form

Use the execute form of the UCBSCAN macro together with the list form for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

Syntax

The execute form of the COPY function of the UCBSCAN macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede UCBSCAN.
UCBSCAN	
b	One or more blanks must follow UCBSCAN.

COPY Default: COPY

,WORKAREA= <i>workarea addr</i>	<i>workarea addr</i> : RX-type address or register (2) - (12).
,UCBAREA= <i>ucbarea addr</i>	<i>ucbarea addr</i> : RX-type address or register (2) - (12).
,CMXTAREA= <i>cmxtarea addr</i> ,CMXTAREA=NONE	<i>cmxtarea addr</i> : RX-type address or register (2) - (12). Default: NONE
,UCBPAREA= <i>ucbparea addr</i> ,UCBPAREA=NONE	<i>ucbparea addr</i> : RX-type address or register (2) - (12). Default: NONE
,DCEAREA= <i>dcearea addr</i> ,DCEAREA=NONE	<i>dcearea addr</i> : RX-type address or register (2) - (12). Default: NONE
,DCELEN= <i>length addr</i>	<i>length addr</i> : RS-type address or register (2) - (12). Note: DCELEN is valid only with DCEAREA and is required with DCEAREA.
,VOLSER= <i>volser addr</i> ,VOLSER=NONE	<i>volser addr</i> : RS-type address or register (2) - (12). Default: NONE
,DEVNCHAR= <i>devnchar addr</i>	<i>devnchar addr</i> : RS-type address or register (2) - (12).
,DEVN= <i>devn addr</i> ,DEVN=0	<i>devn addr</i> : RS-type address or register (2) - (12). Default: 0
,DYNAMIC=NO ,DYNAMIC=YES	Default: NO
,RANGE=3DIGIT ,RANGE=ALL	Default: 3DIGIT
,NONBASE=NO ,NONBASE=YES	Default: NO
,UNBOUND_ALIAS=NO ,UNBOUND_ALIAS=YES ,UNBOUND_ALIAS=ONLY	Default: NO
,DEVCLASS=ALL ,DEVCLASS=CHAR ,DEVCLASS=COMM ,DEVCLASS=CTC ,DEVCLASS=DASD ,DEVCLASS=DISP ,DEVCLASS=TAPE ,DEVCLASS=UREC	Default: ALL
,DEVCID= <i>devcid addr</i> ,DEVCID=0	<i>devcid addr</i> : RS-type address Default: 0
,IOCTOKEN= <i>ioctoken addr</i> ,IOCTOKEN=NONE	<i>ioctoken addr</i> : RX-type address or register (2) - (12). Default: NONE
,LINKAGE=SYSTEM ,LINKAGE=BRANCH	Default: LINKAGE=SYSTEM

UCBSCAN Macro

,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	Default: IMPLIED_VERSION
,PLISTVER= <i>plistver</i>	<i>plistver</i> : 1
,RETCODE= <i>retcode addr</i>	<i>retcode addr</i> : RX-type address or register (2) - (12).
,RSNCODE= <i>rsncode addr</i>	<i>rsncode addr</i> : RX-type address or register (2) - (12).
,MF=(E, <i>list addr</i>)	<i>list addr</i> : RX-type address or register (2) - (12).
,MF=(E, <i>list addr</i> ,COMPLETE)	Default: COMPLETE

Parameters

The parameters are explained under the standard form of the COPY function of the UCBSCAN macro with the following exceptions:

,MF=(E,*list addr*)

,MF=(E,*list addr*,COMPLETE)

Specifies the execute form of the UCBSCAN macro.

The *list addr* parameter specifies the address of the storage area for the parameter list.

COMPLETE specifies that the system is to check for required parameters and supply defaults for optional parameters that were not specified.

UCBSCAN ADDRESS

Syntax

The standard form of the ADDRESS function of the UCBSCAN macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede UCBSCAN.
UCBSCAN	
b	One or more blanks must follow UCBSCAN.

ADDRESS	Note: COPY is the default.
,WORKAREA= <i>workarea addr</i>	<i>workarea addr</i> : RX-type address or register (2) - (12).
,UCBPTR= <i>ucbptr addr</i>	<i>ucbptr addr</i> : RS-type address or register (2) - (12).
,UCBCXPTR= <i>ucbcxptr addr</i>	<i>ucbcxptr</i> : RX-type address or register (2) - (12).

,UCBPXPTR= <i>ucbpxptr addr</i>	<i>ucbpxptr</i> : RX-type address or register (2) - (12).
,LOC=BELOW ,LOC=ANY	Default: BELOW
,PIN ,NOPIN	Note: TEXT and PTOKEN are required with PIN
,TEXT= <i>text addr</i> :	<i>text addr</i> : RX-type address Note: Required with PIN, not valid with NOPIN.
,PTOKEN= <i>p token addr</i> :	<i>p token addr</i> : RS-type address or register (2) - (12). Note: Required with PIN, not valid with NOPIN.
,UCBPAREA= <i>ucbparea addr</i> ,UCBPAREA=NONE	<i>ucbparea addr</i> : RX-type address or register (2) - (12). Default: NONE
,DEVN= <i>devn addr</i> ,DEVN=0	<i>devn addr</i> : RS-type address or register (2) - (12). Default: 0
,DYNAMIC=NO ,DYNAMIC=YES	Default: NO
,RANGE=3DIGIT ,RANGE=ALL	Default: 3DIGIT
,NONBASE=NO ,NONBASE=YES	Default: NO
,UNBOUND_ALIAS=NO ,UNBOUND_ALIAS=YES ,UNBOUND_ALIAS=ONLY	Default: NO
,DEVCLASS=ALL ,DEVCLASS=CHAR ,DEVCLASS=COMM ,DEVCLASS=CTC ,DEVCLASS=DASD ,DEVCLASS=DISP ,DEVCLASS=TAPE ,DEVCLASS=UREC	Default: ALL
,DEVCID= <i>devcid addr</i>	<i>devcid addr</i> : RS-type address
,IOCTOKEN= <i>ioctoken addr</i> ,IOCTOKEN=NONE	<i>ioctoken addr</i> : RX-type address or register (2) - (12). Default: NONE
,LINKAGE=SYSTEM ,LINKAGE=BRANCH	Default: LINKAGE=SYSTEM
,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX ,PLISTVER= <i>plistver</i>	Default: IMPLIED_VERSION <i>plistver</i> : 1 - 2

UCBSCAN Macro

`,RETCODE=retcode addr` *retcode addr*: RX-type address or register (2) - (12).
`,RSNCODE=rsncode addr` *rsncode addr*: RX-type address or register (2) - (12).

Parameters

The parameters are explained as follows:

ADDRESS

Specifies that a UCB address is to be obtained.

,WORKAREA=workarea addr

Specifies the address of a 100-character work area that will be used by the UCBSCAN service. The caller must initialize this work area to binary zeros before starting a UCB scan. On subsequent invocations of UCBSCAN within the same scan, the caller must leave the contents of this work area unchanged.

,UCBPTR=ucbptr addr

Specifies the address of a pointer in which the address of the UCB common segment for the next UCB that meets the search criteria will be returned.

,UCBCXPTR=ucbcxptr addr

Specifies the address of a fullword field in which the system will return the address of the UCB common extension. Use the IEFUCBOB mapping macro to map the UCB common extension segment.

,UCBPXPTR=ucbpxptr addr

Specifies the address of a fullword field in which the system will return the address of the UCB prefix extension. Use the IOSDUPFX mapping macro to map the UCB prefix extension segment.

,LOC=BELOW

,LOC=ANY

Specifies whether the scan should be restricted to below 16 megabyte UCBs (LOC=BELOW) or should also include above 16 megabyte UCBs (LOC=ANY).

,PIN

,NOPIN

Specifies whether the UCB is to be pinned to make it ineligible for deletion through the dynamic UCB process. Pinning the UCB ensures that it will not be deleted while the scan process is taking place. The PIN parameter specifies that the UCB should be pinned, and NOPIN specifies that it should not. Programs that pin a UCB are also responsible for unpinning it once the UCB is no longer subject to processing. Use the UCBPIN macro with the UNPIN option to unpin the UCB.

,TEXT=text addr

Specifies the address of a 58-character input field containing text that documents the reason for the PIN request. If the pin request remains outstanding during a request for a configuration change that would delete this UCB, the text specified by the TEXT parameter will be displayed in a message identifying the reason for a configuration change failure.

,PTOKEN=ptoken addr

Specifies the address of an 8-character area that is to receive the pin token for the UCB. The caller must use the pin token when unpinning the UCB.

,UCBPAREA=ucbparea addr

,UCBPAREA=NONE

Specifies the address of a 48-character storage area that will receive a copy of the UCB prefix extension segment. The area can be mapped by the IOSDUP1 mapping macro.

,DEVN=*devn addr***,DEVN=0**

Specifies (*DEVN=devn addr*) an input halfword that contains, in binary form, the device number with which the scan is to begin. The default, *DEVN=0*, starts the scan with the first UCB.

,DYNAMIC=NO**,DYNAMIC=YES**

Specifies whether the scan should be restricted to static and installation-static UCBs (*DYNAMIC=NO*) or should also include dynamic UCBs (*DYNAMIC=YES*).

,RANGE=3DIGIT**,RANGE=ALL**

Specifies whether the scan should be restricted to UCBs with 3-digit device numbers (*3DIGIT*) or should also include UCBs with 4-digit device numbers (*ALL*).

,NONBASE=NO**,NONBASE=YES**

Specifies whether the scan should include nonbase exposures for a multiple-exposure device, which was supported prior to MVS/ESA SP 5.2. *NO* specifies only the base exposure, and *YES* specifies all exposures.

Specifies whether the scan should include bound alias UCBs for a parallel access volume. *NO* specifies that bound alias UCBs will not be included. *YES* specifies that bound alias UCBs will be included.

,UNBOUND_ALIAS=NO**,UNBOUND_ALIAS=YES****,UNBOUND_ALIAS=ONLY**

Specifies whether the scan should include unbound alias UCBs.

YES Include unbound alias UCBs

NO Do not include unbound alias UCBs

ONLY Include only unbound alias UCBs

Note: The *UNBOUND_ALIAS* function is intended for IOS use only.

,DEVCLASS=ALL**,DEVCLASS=CHAR****,DEVCLASS=COMM****,DEVCLASS=CTC****,DEVCLASS=DASD****,DEVCLASS=DISP****,DEVCLASS=TAPE****,DEVCLASS=UREC**

Specifies the device class that is to be scanned:

ALL Scans UCBs for all device classes

CHAR Scans UCBs for character reader device class

COMM

Scans UCBs for communications device class

CTC Scans UCBs for channel to channel device class

DASD Scans UCBs for direct access device class

DISP Scans UCBs for display device class

UCBSCAN Macro

TAPE Scans UCBs for TAPE device class

UREC Scans UCBs for unit record device class

,DEVCID=devcid addr

Specifies an 8-bit input field used to supply the hexadecimal device class ID of the device class to be scanned. *devcid addr* specifies the address of the field.

If you specify DEVCID, only UCBs of the particular device class specified will be presented, and the DEVCLASS parameter is ignored.

,IOCTOKEN=ioctoken addr

,IOCTOKEN=NONE

Specifies the address of a 48-character area that contains the MVS I/O configuration token that you supply to UCBSCAN. You can obtain this token by issuing the IOCINFO macro, which is described in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*. If the I/O configuration token that is current when UCBSCAN is invoked does not match the token whose address is supplied as input by *ioctoken addr*, the caller will be notified through a return code.

If the input IOCTOKEN (specified by *ioctoken addr*) is set to binary zeros, UCBSCAN will set IOCTOKEN to the current I/O configuration token at the start of the scan.

,LINKAGE=SYSTEM

,LINKAGE=BRANCH

Specifies the type of call that should be generated:

- **SYSTEM:** Specifies a Program Call (PC)
- **BRANCH:** Specifies a Branch entry

LINKAGE=BRANCH is intended for performance-sensitive programs.

,PLISTVER=IMPLIED_VERSION

,PLISTVER=MAX

,PLISTVER=plistver

Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

- 1, if you use only the following parameters:

ADDRESS	LOC	RANGE
DEVCID	MF	RETCODE
DEVCLASS	NONBASE	RSNCODE
DEVN	NOPIN	TEXT
DYNAMIC	PIN	UCBPAREA
IOCTOKEN	PLISTVER	UCBPTR
LINKAGE	PTOKEN	WORKAREA

- 2, if you use any of the following parameters and parameters from *plistver* 1.

UCBCXPTR UCBXPTR

To code, specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1 or 2

,RETCODE=*retcode addr*

Specifies the fullword location where the system is to store the return code. The return code is also in GPR 15.

,RSNCODE=*rsncode addr*

Specifies the fullword location where the system is to store the reason code. The reason code is also in GPR 0.

Return and Reason Codes

When control returns from UCBSCAN, GPR 15 (and *retcode addr*, if you coded RETCODE) contains one of the following return codes:

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	Meaning: UCBSCAN completed successfully. Action: None.
04	01	Meaning: UCBSCAN processing ended. All UCBs that met the search criteria have been presented to the caller. The value stored into the pointer for UCBPTR is unchanged, and the caller-supplied area specified in the WORKAREA parameter has been reset to binary zeros. Action: None.
08	01	Meaning: Program error. A caller in AR mode specified an ALET that was not valid. Action: Correct the ALET and reissue the macro. Possibly the caller wrote over an area in the parameter list; look for this error.
08	02	Meaning: Program error. An error occurred when the system tried to access the caller's parameter list. Action: Ensure that you have met the environmental requirements for the macro, and reissue the macro.

UCBSCAN Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
08	04	Meaning: Program error. An error occurred in referencing the caller-supplied area for the UCB prefix extension segment data. This reason code is valid only for callers using the UCBPAREA parameter. Action: Correct the UCBPAREA parameter.
08	05	Meaning: Program error. An error occurred when the system referenced the caller-supplied area specified in the IOCTOKEN parameter. This reason code is valid only for callers using the IOCTOKEN parameter. Action: Correct the IOCTOKEN parameter.
08	08	Meaning: Program error. An error occurred in referencing the caller-supplied work area specified in the WORKAREA parameter. Action: Correct the WORKAREA parameter.
08	0A	Meaning: Program error. An error occurred in referencing the caller-supplied area for the pin reason text (TEXT). This reason code is valid only for callers using the TEXT parameter. Action: Correct the TEXT parameter.
0C	None	Meaning: Environmental error. The I/O configuration has changed, so that the I/O configuration token supplied through the IOCTOKEN parameter is not current. This return code is valid only for callers using the IOCTOKEN parameter. Action: Obtain the current I/O configuration token by issuing an IOCINFO macro or by setting the input IOCTOKEN parameter in the UCBINFO macro to zero. Start the scan from the beginning.
20	None	Meaning: System error. An unexpected error occurred. Action: Supply the return code to the appropriate IBM support personnel.

UCBSCAN ADDRESS—List Form

Use the list form of the UCBSCAN macro together with the execute form for applications that require reentrant code. The list form of the macro defines an area of storage that the execute form uses for storing the parameters.

Syntax

This macro is an alternative list form macro, and requires a different technique for using the list form as compared to the conventional list form macros. See “Alternative List Form Macros” on page 12 for further information.

The list form of the ADDRESS function of the UCBSCAN macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede UCBSCAN.
UCBSCAN	
b	One or more blanks must follow UCBSCAN.

,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	Default: IMPLIED_VERSION
,PLISTVER= <i>plistver</i>	<i>plistver</i> : 1 - 2
MF=(L, <i>list addr</i>)	<i>list addr</i> : symbol.
MF=(L, <i>list addr</i> , <i>attr</i>)	<i>attr</i> : 1- to 60-character input string.
MF=(L, <i>list addr</i> ,0D)	Default: 0D

Parameters

The parameters are explained under the standard form of the ADDRESS function of the UCBSCAN macro with the following exceptions:

MF=(L,*list addr*)

MF=(L,*list addr*,*attr*)

MF=(L,*list addr*,0D)

Specifies the list form of the UCBSCAN macro.

The *list addr* parameter specifies the address of the storage area for the parameter list.

attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

UCBSCAN ADDRESS—Execute Form

Use the execute form of the UCBSCAN macro together with the list form for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

Syntax

The execute form of the ADDRESS function of the UCBSCAN macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede UCBSCAN.

UCBSCAN Macro

UCBSCAN

b

One or more blanks must follow UCBSCAN.

ADDRESS

Note: COPY is the default.

,WORKAREA=*workarea addr*

workarea addr: RX-type address or register (2) - (12).

,UCBPTR=*ucbptr addr*

ucbptr addr: RS-type address or register (2) - (12).

,UCBCXPTR=*ucbcxptr addr*

ucbcxptr: RX-type address or register (2) - (12).

,UCBPXPTR=*ucbpxptr addr*

ucbpxptr: RX-type address or register (2) - (12).

,LOC=BELOW

Default: BELOW

,LOC=ANY

,PIN

Note: TEXT and PTOKEN are required with PIN

,NOPIN

,TEXT=*text addr*:

text addr: RX-type address

Note: Required with PIN, not valid with NOPIN.

,PTOKEN=*ptoken addr*:

ptoken addr: RS-type address or register (2) - (12).

Note: Required with PIN, not valid with NOPIN.

,UCBPAREA=*ucbparea addr*

ucbparea addr: RX-type address or register (2) - (12).

,UCBPAREA=NONE

Default: NONE

,DEVN=*devn addr*

devn addr: RS-type address or register (2) - (12).

,DEVN=0

Default: 0

,DYNAMIC=NO

Default: NO

,DYNAMIC=YES

,RANGE=3DIGIT

Default: 3DIGIT

,RANGE=ALL

,NONBASE=NO

Default: NO

,NONBASE=YES

,UNBOUND_ALIAS=NO

Default: NO

,UNBOUND_ALIAS=YES

,UNBOUND_ALIAS=ONLY

,DEVCLASS=ALL

Default: ALL

,DEVCLASS=CHAR

,DEVCLASS=COMM

,DEVCLASS=CTC

,DEVCLASS=DASD

,DEVCLASS=DISP

,DEVCLASS=TAPE

,DEVCLASS=UREC	
,DEVCID= <i>devcid addr</i>	<i>devcid addr</i> : RS-type address
,IOCTOKEN= <i>ioctoken addr</i>	<i>ioctoken addr</i> : RX-type address or register (2) - (12).
,IOCTOKEN=NONE	Default: NONE
,LINKAGE=SYSTEM	Default: LINKAGE=SYSTEM
,LINKAGE=BRANCH	
,PLISTVER=IMPLIED_VERSION	Default: IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER= <i>plistver</i>	
	<i>plistver</i> : 1 - 2
,RETCODE= <i>retcode addr</i>	<i>retcode addr</i> : RX-type address or register (2) - (12).
,RSNCODE= <i>rsncode addr</i>	<i>rsncode addr</i> : RX-type address or register (2) - (12).
,MF=(E, <i>list addr</i>)	<i>list addr</i> : RX-type address or register (2) - (12).
,MF=(E, <i>list addr</i> ,COMPLETE)	Default: COMPLETE

Parameters

The parameters are explained under the standard form of the ADDRESS function of the UCBSCAN macro with the following exceptions:

,MF=(E,*list addr*)

,MF=(E,*list addr*,COMPLETE)

Specifies the execute form of the UCBSCAN macro.

The *list addr* parameter specifies the address of the storage area for the parameter list. COMPLETE specifies that the system is to check for required parameters and supply defaults for optional parameters that were not specified.

UCBSCAN Macro

VSMLIST — List Virtual Storage Map

Description

The VSMLIST macro provides information about the allocation of virtual storage. The information is returned in a work area that you specify. The format of the work area is described under “Virtual Storage Management” in *z/OS MVS Programming: Authorized Assembler Services Guide*.

The following information can be requested:

- The ranges of virtual storage allocated to the SQA, by subpool, and the free space within those ranges
- The ranges of virtual storage allocated to the CSA, by subpool, and the free space within those ranges
- The ranges of CSA space that are unallocated
- The ranges of virtual storage allocated to the LSQA in the current address space, by subpool, and the free space within those ranges
- The ranges of virtual storage allocated to private area subpools, by TCB, and the free space within those ranges
- The ranges of private area that are unallocated.

For detailed information about virtual storage subpools, see “Virtual Storage Management” in *z/OS MVS Programming: Authorized Assembler Services Guide*.

Environment

The requirements for the caller are:

Minimum authorization:	For LINKAGE=SYSTEM, problem state and PSW key 8-15. For LINKAGE=BRANCH, supervisor state and PSW key 0.
Dispatchable unit mode:	Task or SRB.
Cross memory mode:	Any PASN, any HASN, any SASN.
AMODE:	24- or 31-bit. All addresses must be 31-bit addresses.
ASC mode:	Primary.
Interrupt status:	Enabled or disabled for I/O and external interrupts.
Locks:	<ul style="list-style-type: none">• For LINKAGE=SYSTEM, the caller may not hold a lock higher than the local lock.• For LINKAGE=BRANCH, see the LINKAGE parameter description for locking requirements.
Control parameters:	Must be in the primary address space. All input parameters, except for the TCB, can reside above 16 megabytes if the caller is running in 31-bit addressing mode. The TCB resides below 16 megabytes.

Programming Requirements

All addresses are associated with the current address space.

You must set bytes 0-3 of the work area to zero before the first invocation of the macro for a given request.

Restrictions

None.

VSMLIST Macro

Input Register Information

Before issuing the VSMLIST macro, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register	Contents
13	Address of a standard 72-byte save area

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14	Used as a work register by the system
15	Return code

When control returns to the caller, the ARs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance Implications

None.

Syntax

The VSMLIST macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
-------------	--

b	One or more blanks must precede VSMLIST.
---	--

VSMLIST

b	One or more blanks must follow VSMLIST.
---	---

SP=SQA

SP=CSA

SP=LSQA

SP=PVT

SP= <i>sp list addr</i>	<i>sp list addr</i> : RX-type address or register (0), (2) - (12)
-------------------------	---

,WKAREA=(<i>addr,length</i>)	<i>addr</i> : RX-type address or register (1) - (12)
--------------------------------	--

	<i>length</i> : Symbol, decimal digit, or register (0), (2) - (12).
--	---

,TCB=(<i>tcb addr</i>) ,TCB=(<i>tcb addr</i> ,ALL) ,TCB=(, ALL)	Default: TCB address in PSATOLD. <i>tcb addr</i> : RX-type address or register (0), (2) - (12). Note: The TCB parameter is required only for SRB routines, if SP=PVT or SP= <i>sp list addr</i> and the list contains private area subpools.
,SPACE=ALLOC ,SPACE=FREE ,SPACE=UNALLOC	Default: SPACE=ALLOC Note: SPACE=UNALLOC can be specified only for SP=CSA or SP=PVT.
,LOC=24 ,LOC=31	Default: LOC=31
,REAL31 ,REAL64	Default: REAL31
,LINKAGE=SYSTEM ,LINKAGE=BRANCH	Default: LINKAGE=SYSTEM
,PVTSP=ALL ,PVTSP=OWNED	Default: PVTSP=ALL

Parameters

The parameters are explained as follows:

SP=SQA

SP=CSA

SP=LSQA

SP=PVT

SP=*sp list addr*

Specifies the storage areas for which information is requested. The following subpools are listed for the specified storage areas:

- SQA: 226, 239, 245, 247, 248
- CSA: 227, 228, 231, 241
- LSQA: 205, 215, 225, 255
- PVT: 0-127, 129-132, 229, 230, 236, 237, 244, 249, 251, 252

GETMAIN/FREEMAIN/STORAGE processing translates the original subpool numbers that were specified on the GETMAIN, FREEMAIN, or STORAGE macros to different subpool numbers as shown below:

Original Subpool Number	Translated Subpool Number
203-205	205
213-215	215
223-225	225
233-235, 253-255	255
0, 240, 250	0

VSMLIST reports the translated subpool numbers, not the original subpool numbers. In addition, VSMLIST does not report incorrect subpool numbers (subpool numbers greater than 255) or undefined subpool numbers.

VSMLIST Macro

If `SP=sp list addr` is specified, the user must supply the address of a subpool list. The first halfword of the list contains the number of entries in the list. Each of the following halfwords in the list contains a subpool number. If a valid subpool number appears more than once in the subpool list, it is reported only once.

,WKAREA=(addr,length)

Indicates the address and length of a user-supplied work area. The system uses this work area to hold the parameter list, control information, and data that is to be returned to the caller. The work area should begin on a word boundary and be a minimum of 4K bytes in length.

You must set bytes 0-3 of this work area to zero before the first invocation of VSMLIST for a specific request. See "Virtual Storage Management" in *z/OS MVS Programming: Authorized Assembler Services Guide* for a description of the work area.

,TCB=(tcb addr)

,TCB=(tcb addr,ALL)

,TCB=(,ALL)

Specifies the TCB associated with the virtual storage allocated to the private area subpools. The TCB must be located in the currently addressable address space. If ALL is specified, the storage associated with the TCB and all of its subtasks is reported.

Notes:

1. If ALL is specified and the TCB is high in the task structure (for example, the TCB for RCT), more than one region could be listed. The regions in the private area are the RCT region, the V=V region, and the V=R region (for V=R jobs).
2. The TCB resides in storage below 16 megabytes.

,SPACE=ALLOC

,SPACE=FREE

,SPACE=UNALLOC

Specifies whether allocated, allocated and free, or unallocated storage is to be reported.

ALLOC indicates that the virtual addresses and lengths of blocks of storage allocated to the specific area are to be listed.

FREE indicates that in addition to the information supplied by ALLOC, the virtual addresses and lengths of free space within the allocated blocks are to be listed.

UNALLOC indicates that the virtual addresses and lengths of unallocated blocks of storage are to be listed. Both TCB and REAL are ignored when UNALLOC is specified.

Note: An allocated block of storage is a block that is a multiple of 4K in size and contains some storage that has been allocated via a GETMAIN or STORAGE macro. The free storage is the storage within an allocated block that has not been allocated via a GETMAIN or STORAGE macro. An unallocated block of storage is a block that is a multiple of 4K in size and contains no allocated storage.

,LOC=24

,LOC=31

Indicates whether information should be returned about virtual storage areas residing above and below 16 megabytes or only those residing below 16

megabytes. If LOC=31 is specified, information is returned for all storage areas below 2 gigabytes. If LOC=24 is specified, information is returned only for storage areas below 16 megabytes.

Note: Specifying LOC=BELOW is the same as specifying LOC=24. Specifying LOC=ANY is the same as specifying LOC=31. The old values are still supported, but IBM recommends using the newer values instead.

,REAL31

,REAL64

Indicates that the high order bit of the address field of the allocated block descriptor should be set to show the value specified on the LOC parameter of the GETMAIN, STORAGE, or CPOOL macro invocation used to obtain that storage area. If the storage block was allocated using any LOC specification of GETMAIN or STORAGE except LOC=(,24), the indicator is turned on; if the storage block was allocated using the LOC=(,24) parameter of the GETMAIN or STORAGE macros, the indicator is turned off.

When REAL31 is specified

If the storage block is backed in real 31-bit or 64-bit storage, the high bit indicator is on (one). If the storage block is backed in real 24-bit storage, the high bit indicator is off (zero). The low bit indicator is always off.

When REAL64 is specified

If the storage block is backed in real 64-bit storage, the low bit indicator is on (one). If the storage block is backed in real 31-bit storage, the high bit indicator is on (one). If the storage block is backed in real 24-bit storage, both indicators are off.

When neither is specified

Both indicators remain off (zero).

Note: The REAL parameter is deprecated, but still supported by VSMLIST. It has the same function as REAL31. IBM recommends using REAL31 instead.

,LINKAGE=SYSTEM

,LINKAGE=BRANCH

Indicates whether the VSMLIST routine uses a PC instruction (LINKAGE=SYSTEM) or branch entry (LINKAGE=BRANCH) for linkage and whether the VSMLIST routine provides serialization and recovery.

If LINKAGE=SYSTEM is specified, the VSMLIST routine provides linkage using a PC instruction and also provides recovery and serialization.

The caller's secondary ASID is preserved when a PC is issued.

Note: Serialization is not provided across calls to VSMLIST.

If LINKAGE=BRANCH is specified, the VSMLIST routine uses branch entry for linkage and does not provide recovery or serialization. Before issuing VSMLIST, provide serialization as follows:

- For LSQA or PVT requests, obtain the LOCAL lock.
- For SQA or CSA requests, issue the SETLOCK macro right before and right after the VSMLIST request, as follows:

```
SETLOCK OBTAIN,TYPE=VSMFIX,MODE=UNCOND
VSMLIST request
SETLOCK RELEASE,TYPE=VSMFIX
```

VSMLIST Macro

If your program is covered by a functional recovery routine (FRR) and the FRR receives control after SETLOCK OBTAIN has been issued and before SETLOCK RELEASE has been issued, your FRR must either issue SETLOCK RELEASE,TYPE=VSMFIX or must issue the SETLOCK and SETRP macros as follows:

```
        SETLOCK TEST,TYPE=VSMFIX,BRANCH=(NOTHELD,NOVSMFIX)
        SETRP FRELOCK=VSMFIX
NOVSMFIX DS 0H
```

Notes:

1. Your program will be disabled for I/O and external interrupts from the time the SETLOCK OBTAIN completes until the SETLOCK RELEASE completes. See the descriptions of the SETLOCK and SETRP macros for additional usage information.
2. Your program and your FRR must not issue any of the following macros before issuing SETLOCK RELEASE,TYPE=VSMFIX:
 - GETMAIN
 - FREEMAIN
 - STORAGE

,PVTSP=ALL

,PVTSP=OWNED

Indicates for each task, which subpool information will be returned.

ALL indicates that information is returned for subpools which are owned or shared by the task.

OWNED indicates that information is returned only for the subpools which are owned by the task.

ABEND Codes

The VSMLIST macro might issue abend code X'C78'. For detailed abend code information, see *z/OS MVS System Codes*.

Return and Reason Codes

When the VSMLIST macro returns control to your program, GPR 15 contains one of the following hexadecimal return codes:

Table 39. Return Codes for the VSMLIST Macro

Return Code	Meaning and Action
0	Meaning: Successful completion. Action: None.
4	Meaning: Partially successful completion. More information remains to be returned in the work area. Action: Reissue the macro to obtain additional information until a return code of 0 is returned in register 15. Do not change the value of bytes 0-3 of the work area before reissuing the macro.

Table 39. Return Codes for the VSMLIST Macro (continued)

Return Code	Meaning and Action
8	<p>Meaning: System error. The system encountered an error while scanning virtual storage management data areas. The information in the data area is valid, but incomplete. This return code is obtained only by users who specify LINKAGE=SYSTEM.</p> <p>Action: Notify system support personnel that there might be an error in virtual storage management's control structure. If there is a problem with VSM's control structure, the entire system will be adversely affected and you might need to wait until the problem is identified and resolved by support personnel. Support personnel should take a dump of the virtual storage management control structure to help identify the cause of the problem. If the problem appears to involve common storage, the contents of common storage should be dumped to view the VSM control structure. If the problem appears to involve private storage, private storage should be dumped.</p>
C	<p>Meaning: Program error. The system detected one of the following errors:</p> <ul style="list-style-type: none"> • The work area was too small. • An incorrect parameter was specified. • Incorrect control information was in the work area. <p>This return code is obtained only by users who specify LINKAGE=BRANCH. Users who specify LINKAGE=SYSTEM receive a X'C78' abend for these errors.</p> <p>Action: Ensure that the work area is at least 4096 bytes long. Verify that the work area is correctly defined and initialized and that parameters are specified properly. Verify that your program has not inadvertently modified the VSMLIST work area.</p>

Example 1

List the ranges of the allocated and free storage in the SQA. Specify the address of the VSM work area in register 2 and the length of the work area in register 3.

```
VSMLIST SP=SQA,SPACE=FREE,WKAREA=((2),(3))
```

Example 2

List the ranges of the allocated storage in the CSA. Specify the address of the work area in register 2 and the length of the work area in register 3. Provide branch entry linkage.

```
VSMLIST SP=CSA,SPACE=ALLOC,WKAREA=((2),(3)),LINKAGE=BRANCH
```

Example 3

List the ranges of unallocated storage in the private area. The variable X contains the address of the work area, which has a length of 4096 bytes.

```
VSMLIST SP=PVT,SPACE=UNALLOC,WKAREA=(X,4096)
```

Example 4

List the ranges of allocated storage, below 16 megabytes, in each of the subpools specified in the subpool list at location Y. The variable X contains the address of the work area, which has a length of 4096 bytes.

```
VSMLIST SP=Y,SPACE=ALLOC,WKAREA=(X,4096),LOC=BELOW
```

VSMLIST Macro

VSMLOC — Verify Virtual Storage Allocation

Description

The VSMLOC macro verifies that a given storage area has been allocated using the GETMAIN or STORAGE macros.

Environment

The requirements for the caller are:

Minimum authorization:	For LINKAGE=SYSTEM, problem state and PSW key 8-15. For LINKAGE=BRANCH, supervisor state and PSW key 0.
Dispatchable unit mode:	Task or SRB.
Cross memory mode:	Any PASN, any HASN, any SASN.
AMODE:	24- or 31-bit. All addresses passed to VSMLOC must be 31-bit addresses.
ASC mode:	Primary.
Interrupt status:	For LINKAGE=SYSTEM, enabled or disabled for I/O and external interrupts. For LINKAGE=BRANCH, interrupt status depends on the area of storage for which information is requested. See the LINKAGE=BRANCH parameter for complete information.
Locks:	<ul style="list-style-type: none">• For LINKAGE=SYSTEM with LSQA, PVT, or CPOOLLCL specified, you may hold only the local lock.• For LINKAGE=BRANCH, see the LINKAGE=BRANCH parameter for locking requirements.
Control parameters:	Must be in the primary address space

Programming Requirements

- All addresses are associated with the current address space.
- The VSMLOC service does not provide serialization or recovery for callers specifying LINKAGE=BRANCH. Callers must provide serialization and recovery as described under the LINKAGE parameter description.
- The VSMLOC service provides serialization and recovery for callers specifying LINKAGE=SYSTEM.

Restrictions

None.

Input Register Information for LINKAGE=SYSTEM

Before issuing the VSMLOC macro with LINKAGE=SYSTEM, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register	Contents
13	Address of a standard 72-byte save area

Output Register Information for LINKAGE=SYSTEM

When control returns to the caller, the GPRs contain:

Register	Contents
0	If GPR15 contains a value of zero, byte 3 contains the number of

VSMLOC Macro

the subpool from which the specified storage area was obtained.
Bytes 0-2 do not contain any relevant information.

	If GPR15 contains a nonzero value, contains zero.
1	Used as a work register by the system.
2-13	Unchanged.
14	Used as a work register by the system.
15	Return code.

When control returns to the caller, the access registers (ARs) contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

Input Register Information for LINKAGE=BRANCH

Before issuing the VSMLOC macro with LINKAGE=BRANCH, the caller must ensure that the following general purpose registers (GPRs) contain the following information:

Register	Contents
13	Address of a standard 72-byte save area

Output Register Information for LINKAGE=BRANCH

When control returns to the caller, the GPRs contain:

Register	Contents
0	If GPR15 contains a value of 0, byte 3 contains the number of the subpool from which the specified storage area was obtained. Bytes 0-2 do not contain any relevant information.
	If GPR15 contains a nonzero value, contains zero.
1-2	Used as work registers by the system
3-13	Unchanged.
14	Used as a work register by the system.
15	Return code.

When control returns to the caller, the ARs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance Implications

None.

Syntax

The VSMLOC macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede VSMLOC.
VSMLOC	
b	One or more blanks must follow VSMLOC.
<hr/>	
SQA	
CSA	
LSQA	
PVT	
CPOOLFIX	
CPOOLPAG	
CPOOLLCL	
,AREA=(<i>addr,length</i>)	<i>addr</i> : RX-type address or register (0) - (12). <i>length</i> : Symbol, decimal digit or register (0), (2) - (12). Use only with SQA, CSA, LSQA, and PVT.
,AREA=(<i>addr</i>)	<i>addr</i> : RX-type address or register (0) - (12). Use only with CPOOLFIX, CPOOLPAG, and CPOOLLCL.
,TCB= <i>addr</i>	<i>addr</i> : RX-type address or register (0) - (12). Can only be specified with PVT.
,LINKAGE=SYSTEM	Default: LINKAGE=SYSTEM
,LINKAGE=BRANCH	

Parameters

The parameters are explained as follows:

SQA
CSA
LSQA
PVT
CPOOLFIX
CPOOLPAG
CPOOLLCL

Used to verify that storage has been allocated.

SQA, CSA, LSQA, and PVT are used to verify that storage for SQA, CSA, LSQA, or PVT (private area storage) has been allocated in the current address space.

CPOOLFIX is used to verify that storage for a global fixed cell pool has been allocated. Users who obtain their storage from subpool 226, 227, 228, 239, or 245 should specify this keyword.

VSMLOC Macro

CPOOLPAG is used to verify that storage for a global pageable cell pool has been allocated. Users who obtain storage from subpool 231, 241, 247, or 248 should specify this keyword.

CPOOLLCL is used to verify that storage for a local cell pool has been allocated. Users who obtain storage from subpool 0-127, 129-132, 203-205, 213-215, 223-225, 229, 230, 233-237, 240, 249, or 250-255 should specify this keyword.

,AREA=(*addr,length*)

Indicates the start of the virtual storage area (*addr*) and the length of the virtual storage area (*length*) to be verified.

,AREA=(*addr*)

Indicates the start of the virtual storage area (*addr*) to be verified.

,TCB=*addr*

Indicates that VSMLOC is to place the address of the TCB associated with the verified storage in the register or storage area specified by the TCB parameter. If the return code from VSMLOC is not zero, the register or storage area specified by the TCB parameter is set to zero. The TCB parameter can be specified only with PVT.

,LINKAGE=SYSTEM

,LINKAGE=BRANCH

Indicates the type of linkage that VSMLOC is to use.

If LINKAGE=SYSTEM is specified, the VSMLOC routine uses a basic PC instruction for linkage.

The caller's secondary ASID is preserved when a basic PC is issued.

If LINKAGE=BRANCH is specified, the VSMLOC routine uses branch entry linkage. Before issuing VSMLOC, provide serialization as follows:

- For LSQA, CPOOLLCL, and PVT requests, obtain the LOCAL lock.
- For CSA, SQA, and CPOOLFIX requests, issue the SETLOCK macro right before and right after the VSMLOC request, as follows:

```
SETLOCK OBTAIN,TYPE=VSMFIX,MODE=UNCOND
VSMLOC request
SETLOCK RELEASE,TYPE=VSMFIX
```

If your program is covered by a functional recovery routine (FRR) and the FRR receives control after SETLOCK OBTAIN has been issued and before SETLOCK RELEASE has been issued, your FRR must either issue SETLOCK RELEASE,TYPE=VSMFIX or must issue the SETLOCK and SETRP macros as follows:

```
SETLOCK TEST,TYPE=VSMFIX,BRANCH=(NOTHELD,NOVSMFIX)
SETRP FRELOCK=VSMFIX
NOVSMFIX DS 0H
```

Notes:

1. Your program will be disabled for I/O and external interrupts from the time the SETLOCK OBTAIN completes until the SETLOCK RELEASE completes. See the descriptions of the SETLOCK and SETRP macros for additional usage information.
2. Your program and your FRR must not issue any of the following macros before issuing SETLOCK RELEASE,TYPE=VSMFIX:
 - GETMAIN
 - FREEMAIN
 - STORAGE

- For CPOOLPAG requests, issue the SETLOCK macro right before and right after the VSMLOC request, as follows:

```
SETLOCK OBTAIN,TYPE=VSMPAG,MODE=UNCOND
VSMLOC request
SETLOCK RELEASE,TYPE=VSMPAG
```

If your program is covered by a functional recovery routine (FRR) and the FRR receives control after SETLOCK OBTAIN has been issued and before SETLOCK RELEASE has been issued, your FRR must either issue SETLOCK RELEASE,TYPE=VSMPAG or must issue SETLOCK and SETRP macros as follows:

```
SETLOCK TEST,TYPE=VSMPAG,BRANCH=(NOTHELD,NOVSMPAG)
SETRP FRELOCK=VSMPAG
NOVSMPAG DS 0H
```

Notes:

- Your program will be disabled for I/O and external interrupts from the time the SETLOCK OBTAIN completes until the SETLOCK RELEASE completes. See the descriptions of the SETLOCK and SETRP macros for additional usage information.
- Your program and your FRR must not issue any of the following macros before issuing SETLOCK RELEASE,TYPE=VSMPAG:
 - GETMAIN
 - FREEMAIN
 - STORAGE

ABEND Codes

The VSMLOC macro might issue abend code X'C78'. For detailed abend code information, see *z/OS MVS System Codes*.

Return and Reason Codes

When the VSMLOC macro returns control to your program, GPR 15 contains one of the following hexadecimal return codes:

Table 40. Return Codes for the VSMLOC Macro

Return Code	Meaning and Action
0	<p>Meaning: Successful completion. The specified virtual storage area is allocated.</p> <p>Action: None.</p>
4	<p>Meaning: Possible program error. The specified virtual storage area is:</p> <ul style="list-style-type: none"> Not allocated. Overlaps free space. Overlaps other subpools. <p>Action: This return code is not a program error if you have issued VSMLOC to determine whether the storage at the specified address is currently allocated and the system indicates that it is not. This return code signifies a program error if you expected the specified storage area to be allocated and the system reports one of the conditions listed above. If this is an error, you need to determine why the storage area is in the indicated state. Possible reasons include:</p> <ul style="list-style-type: none"> The storage area address or length is not valid. The storage has been freed by another program. The storage was in a subpool that is automatically freed by the system and the system has freed the storage.

VSMLOC Macro

Table 40. Return Codes for the VSMLOC Macro (continued)

Return Code	Meaning and Action
8	<p>Meaning: System error. The system encountered an error while scanning virtual storage management data areas. This return code is obtained only by users who specify LINKAGE=SYSTEM.</p> <p>Action: Notify system support personnel that there might be an error in virtual storage management's control structure. If there is a problem with VSM's control structure, the entire system will be adversely affected and you might have to wait until the problem is identified and resolved. System support personnel should request a dump of the virtual storage management control structure and contact IBM support. If the problem appears to involve common storage, the contents of common storage should be dumped to view the VSM control structure. If the problem appears to involve private storage, private storage should be dumped.</p>
C	<p>Meaning: Program error. You have specified a parameter incorrectly. This return code is obtained only by users who specify LINKAGE=BRANCH. Users who specify LINKAGE=SYSTEM receive a X'C78' abend for this error.</p> <p>Action: Ensure that the virtual storage area you have specified does not exceed 2 gigabytes. Verify that you have coded the parameters as required.</p>
10	<p>Meaning: System error. Internal system error.</p> <p>Action: Record the return code and notify IBM support personnel.</p>

Example 1

Verify that the virtual storage, starting at the address given in register 2 and having a length specified in register 3, has been allocated in the SQA.

```
VSMLOC SQA,AREA=((2),(3))
```

Example 2

Verify that the 8-bytes of virtual storage starting at X have been allocated in the CSA. Use a PC instruction for linkage and let VSMLOC provide recovery and serialization.

```
VSMLOC CSA,AREA=(X,8),LINKAGE=SYSTEM
```

Example 3

Verify that the 8-bytes of virtual storage starting at the address specified in register 2 have been allocated in the LSQA. Use branch entry for linkage.

```
VSMLOC LSQA,AREA=((2),8),LINKAGE=BRANCH
```

Example 4

Verify that the virtual storage, starting at X and having a length specified in register 3, has been allocated in private area storage. Use branch entry for linkage.

```
VSMLOC PVT,AREA=(X,(3)),LINKAGE=BRANCH
```

Example 5

Verify that the 100 bytes of virtual storage starting at the address specified in register 1 have been allocated in private area storage. The address of the TCB associated with the storage verified is returned in register 4.

```
VSMLOC PVT,AREA=((1),100),TCB=(4),LINKAGE=BRANCH
```

VSMREGN — Obtain Private Area Region Size

Description

The VSMREGN macro provides the virtual starting address and sizes of the private area user regions associated with a given TCB in the current address space. For more information about the user region, see *z/OS MVS Initialization and Tuning Guide*.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state and PSW key 8-15.
Dispatchable unit mode:	Task or SRB.
Cross memory mode:	Any PASN, any HASN, any SASN.
AMODE:	24- or 31-bit. All addresses passed to VSMREGN must be 31-bit addresses.
ASC mode:	Primary.
Interrupt status:	Enabled or disabled for I/O and external interrupts.
Locks:	<ul style="list-style-type: none">• If obtaining user region information for the currently active task by specifying its TCB address or by taking the default value for the TCB keyword, the caller is not required to hold any locks.• Otherwise, the caller must hold the local lock of the currently addressable address space.
Control parameters:	Must be in the primary address space. All input parameters except for the TCB address can reside above 16 megabytes if the caller is running in 31-bit addressing mode. The TCB resides below 16 megabytes.

Programming Requirements

None.

Restrictions

None.

Input Register Information

Before issuing the VSMREGN macro, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register	Contents
13	Address of a standard 72-byte save area

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14	Used as a work register by the system
15	Return code

VSMREGN Macro

When control returns to the caller, the ARs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance Implications

None.

Syntax

The VSMREGN macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
<i>b</i>	One or more blanks must precede VSMREGN.
VSMREGN	
<i>b</i>	One or more blanks must follow VSMREGN.

WKAREA= <i>addr</i>	<i>addr</i> : RX-type address or register (0) - (12).
<i>,TCB=tcb addr</i>	<i>tcb addr</i> : RX-type address or register (0), (2) - (12). Default: (except for SRB routines) TCB address in PSATOLD.

Parameters

The parameters are explained as follows:

WKAREA=*addr*

Indicates the virtual address of a 16-byte work area, which is used by VSMREGN to return the requested information. The format of the work area is:

Bytes Meaning

0-3	Virtual address of the region below 16 megabytes
4-7	Length of the region below 16 megabytes
8-11	Virtual address of the region above 16 megabytes
12-15	Length of the region above 16 megabytes

,TCB=*tcb addr*

Indicates the virtual address of the TCB to be used to identify the region (the region control task (RCT) region, the V=V region, or the V=R region). SRB

routines and routines whose currently addressable address space is not the home address space must specify the TCB operand. They cannot use the default value.

ABEND Codes

None.

Return and Reason Codes

When control returns from VSMREGN, GPR 15 always contains a return code of zero, indicating successful completion.

Example 1

Find the virtual address and length of the private area of the TCB whose address is in PSATOLD. Return the information in the work area whose address is given in register 2.

VSMREGN WKAREA=(2)

Example 2

Find the virtual address and length of the private area of the TCB specified in register 3. Return this information in the work area whose address is given in register 2.

VSMREGN WKAREA=(2),TCB=(3)

Example 3

Find the virtual address and length of the private area of the TCB whose address is X. Return this information in the work area whose address is given in register 2.

VSMREGN WKAREA=(2),TCB=X

Example 4

Find the virtual address and length of the private area of the TCB whose address is given in register 3. Return this information in the work area whose address is X.

VSMREGN WKAREA=X,TCB=(3)

VSMREGN Macro

WAIT — Wait for One or More Events

Description

The WAIT macro is used to tell the system that performance of the active task cannot continue until one or more specific events, each represented by a different event control block (ECB), have occurred. Bit 0 and bit 1 of each ECB must be set to zero before it is used.

The system takes the following action:

- For each event that has already occurred (each ECB is already posted), the count of the number of events is decreased by one.
- If the number of events is zero by the time the last event control block is checked, control is returned to the instruction following the WAIT macro.
- If the number of events is not zero by the time the last ECB is checked, control is not returned to the issuing program until sufficient ECBs are posted to bring the number to zero. Control is then returned to the instruction following the WAIT macro.

See *z/OS MVS Programming: Authorized Assembler Services Guide* for information on how to use the WAIT macro to serialize resources.

Environment

The requirements for callers of WAIT are:

Minimum authorization:	Supervisor state or problem state, with any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	One of the following: <ul style="list-style-type: none">• For LINKAGE=SVC: PASN=HASN=SASN• For LINKAGE=SYSTEM: PASN=HASN=SASN or PASN=HASN=SASN
AMODE:	24- or 31- or 64-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interruptions
Locks:	<ul style="list-style-type: none">• For LINKAGE=SYSTEM: No locks held• For LINKAGE=SVC: No locks held, and no enabled unlocked task (EUT) functional recovery routines (FRR) established
Control parameters:	ECB and ECBLIST must be in the home address space.

Programming Requirements

None.

Restrictions

When using LINKAGE=SVC (the default), the caller cannot have an EUT FRR established.

Input Register Information

Before issuing the WAIT macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

WAIT Macro

Output Register Information

After the caller issues the macro, the system might use some registers as work registers or might change the contents of some registers. When the system returns control to the caller, the contents of these registers are not the same as they were before the macro was issued. Therefore, if the caller depends on these registers containing the same value before and after issuing the macro, the caller must save these registers before issuing the macro and restore them after the system returns control.

When control returns to the caller, the general purpose registers (GPRs) contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	One of the following: <ul style="list-style-type: none">For LINKAGE=SYSTEM: Used as work registers by the systemFor LINKAGE=SVC: Unchanged

When control returns to the caller, the access registers (AR) contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

Performance Implications

None.

Syntax

The WAIT macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede WAIT.
WAIT	
b	One or more blanks must follow WAIT.
<i>event nmbr</i> ,	<i>event nmbr</i> : Symbol, decimal digit, or register (0) or (2) - (12). Default: 1 Value range: 0-255
ECB= <i>ecb addr</i>	<i>ecb addr</i> : RX-type address, or register (1) or (2) - (12).
ECBLIST= <i>ecb list addr</i>	<i>ecb list addr</i> : RX-type address, or register (1) or (2) - (12).
,LONG=NO ,LONG=YES	Default: LONG=NO

<code>,LINKAGE=SVC</code> <code>,LINKAGE=SYSTEM</code>	Default: LINKAGE=SVC
<code>,EUT=NOSAVE</code> <code>,EUT=SAVE</code>	Default: EUT=NOSAVE
<code>,RELATED=value</code>	<i>value:</i> Any valid macro keyword specification.

Parameters

The parameters are explained as follows:

event nmbr,

Specifies the number of events waiting to occur.

ECB=*ecb addr*

ECBLIST=*ecb list addr*

Specifies the address of an ECB on a fullword boundary or the address of a virtual storage area containing one or more consecutive fullwords on a fullword boundary. Each fullword contains the address of an ECB; the high order bit in the last fullword must be set to one to indicate the end of the list.

The ECB parameter is valid only if the number of events is specified as one or is omitted. The number of ECBs in the list specified by the ECBLIST form must be equal to or greater than the specified number of events.

If you specify ECBLIST, *ecb list addr* and all ECBs on the list must be in the home address space.

,LONG=NO

,LONG=YES

Specifies whether the task is entering a long wait (YES) or a regular wait (NO).

,LINKAGE=SVC

,LINKAGE=SYSTEM

Specifies whether the caller is in cross memory mode (LINKAGE=SYSTEM) or not (LINKAGE=SVC).

When the caller is not in cross memory mode (the primary, secondary, and home address spaces are the same), use LINKAGE=SVC. With this parameter, linkage is through an SVC instruction.

When the caller is in cross memory mode (the primary, secondary, and home address spaces are not the same), use LINKAGE=SYSTEM. With this parameter, linkage is through a PC instruction. Note that the ECB must be in the home address space.

,EUT=NOSAVE

,EUT=SAVE

Specifies whether enabled unlocked task (EUT) FRRs, if present, should be preserved around the WAIT processing. Specify this keyword only if you specify LINKAGE=SYSTEM.

,RELATED=value

Specifies information used to self-document macros by “relating” functions or

WAIT Macro

services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user, and may be any valid coding values.

The RELATED parameter is available on macros that provide opposite services (for example, ATTACH/DETACH, GETMAIN/FREEMAIN, and LOAD/DELETE), and on macros that relate to previous occurrences of the same macros (for example, CHAP and ESTAE).

The RELATED parameter may be used, for example, as follows:

```
WAIT1 WAIT 1,ECB=ECB,RELATED=(RESUME1,
                                'WAIT FOR EVENT')
.
.
.
RESUME1 POST ECB,0,RELATED=(WAIT1,
                              'RESUME WAITER')
```

Note: Each of these macros will fit on one line when coded, so there is no need for a continuation indicator.

CAUTION:

A job step with all of its tasks in a WAIT condition is terminated upon expiration of the time limits that apply to it.

Example

You have previously initiated one or more activities to be completed asynchronously to your processing. As each activity was initiated, you set up an ECB in which bits 0 and 1 were set to zero. You now wish to suspend your task via the WAIT macro until a specified number of these activities have been completed.

Completion of each activity must be made known to the system via the POST macro. POST causes an addressed ECB to be marked complete. If completion of the event satisfies the requirements of an outstanding WAIT, the waiting task is marked ready and will be executed when its priority allows.

ABEND Codes

The caller of WAIT might encounter one of the following abend codes:

```
101
201
301
401
```

See *z/OS MVS System Codes* for explanations and responses for these codes.

Return and Reason Codes

None.

Example 1

Wait for one event to occur (with a default count).

```
WAIT ECB=WAITECB
.
.
WAITECB DC F'0'
```

Example 2

Wait for two events to occur.

```

        WAIT    2,ECBLIST=LISTECBS
        .
        .
LISTECBS DC  A(ECB1)
        DC  A(ECB2)
        DC  X'80'
        DC  AL3(ECB3)

```

Example 3

Enter a long wait for a task.

```

        WAIT    1,ECBLIST=LISTECBS, LONG=YES
        .
        .
        .
LISTECBS DC  A(ECB1)
        DC  A(ECB2)
        DC  X'80'
        DC  AL3(ECB3)

```

WAIT Macro

WTL — Write To Log

Description

Note: IBM recommends that you use the WTO macro with the MCSFLAG=HRDCPY parameter instead of WTL, because WTO supplies more data than WTL.

The WTL macro causes a message to be written to the system log (SYSLOG) or the operations log (OPERLOG) log stream depending on which one of these logs, or both, is active. The message can include any character that can be used in a C-type (character) DC statement, and is assembled as a variable-length record.

Note: When a message is recorded in SYSLOG, the exact format of the output of the WTL macro varies depending on the job entry subsystem (JES2 or JES3) that is being used, the output class that is assigned to the log at system initialization, and whether DLOG is in effect for JES3. See the *z/OS MVS System Messages* manuals for information on the format of logged messages.

z/OS JES3 Commands also contains information about the format of logged messages.

The description of the WTL macro follows. The WTL macro is also described in *z/OS MVS Programming: Assembler Services Reference ABE-HSP* (with the exception of the OPTION parameter).

Environment

The requirements for the caller are:

Minimum authorization:	Problem state and any PSW key. For OPTION, APF-authorized with PSW key 0-7, or supervisor state
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in the primary address space

Programming Requirements

None.

Restrictions

Message text cannot exceed 126 characters. If the message text exceeds 126 characters, truncation occurs at the last embedded blank before the 126th character; when there are no embedded blanks, truncation occurs after the 126th character.

WTL Macro

Input Register Information

Before issuing the WTL macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0	Reason code
1-14	Unchanged
15	Return code

When control returns to the caller, the access registers (ARs) contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance Implications

None.

Syntax

The standard form of the WTL macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
␣	One or more blanks must precede WTL.
WTL	
␣	One or more blanks must follow WTL.

<i>'msg'</i>	<i>msg</i> : Up to 126 characters if OPTION=NOPREFIX is specified. Up to 128 characters if OPTION=PREFIX is specified.
,OPTION=PREFIX ,OPTION=NOPREFIX	Default: OPTION=NOPREFIX

Parameters

The parameters are explained as follows:

'msg'

Specifies the message to be written to the system log and/or the operations log. The message must be enclosed in apostrophes, which will not appear in the log. The message can include any character that can be used in a C-type (character) DC statement, and is assembled as a variable-length record. See "Timing and Communication" in *z/OS MVS Programming: Assembler Services Guide* for a list of the printable EBCDIC characters passed to display devices or printers.

,OPTION=PREFIX

,OPTION=NOPREFIX

Specifies whether the WTL text contains a prefix identifying the system log record. If PREFIX is specified, the text already contains a prefix. If NOPREFIX is specified or if this parameter is omitted, a two-character prefix will be added by the system. The OPTION keyword is ignored by any program running in the JES3 primary address space.

ABEND Codes

None.

Return and Reason Codes

When the WTL macro returns control to your program, GPR 15 contains a hexadecimal return code and GPR 0 contains a hexadecimal reason code. WTL issues a return code (either 00 or 04), with multiple reason codes for each. The return codes indicate the following:

- 00 - WTL wrote the message to the system log, the operations log, or both.
- 04 - WTL could not write the message to either the system log or the operations log.

Table 41. Return and Reason Codes for the WTL Macro

Return Code	Reason Code	Meaning and Action
00	00	<p>Meaning: WTL processing completed successfully. The system logged the message in SYSLOG, and if OPERLOG was requested, the system logged the message in OPERLOG.</p> <p>Action: None.</p>
00	04	<p>Meaning: WTL processing completed successfully. The message was logged in the operations log (OPERLOG logstream). The system log was not active.</p> <p>Action: If you want the message logged in the system log, start the system log and rerun the program.</p>
00	08	<p>Meaning: WTL processing completed, but the message was only logged in the operations log because the WTL system log buffers are full.</p> <p>Action: Do one of the following, if you want subsequent messages logged in the system log:</p> <ul style="list-style-type: none"> • Enter a CONTROL M,LOGLIM command to change the allocated number of WTL system log buffers dynamically. • Change the LOGLIM value specifying the number of WTL system log buffers on the INIT statement in the CONSOLxx parmlib member. This value will take effect at the next IPL.

Table 41. Return and Reason Codes for the WTL Macro (continued)

Return Code	Reason Code	Meaning and Action
00	0C	<p>Meaning: WTL processing completed, but the message was only logged in the system log because the operations log was not active.</p> <p>Action: If you want the message logged in the operations log, start the operations log and rerun the program. This will also place the message in the system log.</p>
00	10	<p>Meaning: WTL processing completed, but the message was only logged in the system log. The message was not logged in the operations log stream because of a storage problem.</p> <p>Action: If you want the message logged in the operations log, retry the request. This will also place the message in the system log. If the problem persists, contact the IBM Support Center. Provide the return and reason code.</p>
04	04	<p>Meaning: System error. WTL processing was not successful. Recovery could not be established.</p> <p>Action: Retry the request. If the problem persists, record the return and reason code and supply it to the appropriate IBM support personnel.</p>
04	08	<p>Meaning: Environmental error. The system log and the operations log are not active.</p> <p>Action: Start the logs and rerun your program.</p>
04	0C	<p>Meaning: Environmental error. The WTL limit has been reached.</p> <p>Action: Do one of the following:</p> <ol style="list-style-type: none"> 1. Retry the request when the shortage is relieved. 2. Issue a CONTROL M,LOGLIM command to change the allocated number of WTL SYSLOG buffers. 3. Change the LOGLIM value on the INIT statement in the CONSOLxx member of SYS1.PARMLIB. This new value will take effect at the next IPL. <p>Note: If the problem is persistent, you might want to perform step 2 first and step 3 at the next IPL.</p>
04	10	<p>Meaning: System error. An internal error occurred. The system issues message IEE390I.</p> <p>Action: Contact the IBM Support Center. Provide the return and reason code.</p>
04	14	<p>Meaning: System error. The system encountered a (VSM) error. The system issues message IEE390I.</p> <p>Action: Contact the IBM Support Center. Provide the return and reason code.</p>
04	18	<p>Meaning: Environmental error. The message was not logged in either the system log or the operations log, because neither log is active.</p> <p>Action: Do one of the following:</p> <ul style="list-style-type: none"> • If you want to log the message in the operations log, start the operations log with the VARY OPERLOG,HARDCPY command and rerun the program. • If you want the message logged in the system log, start the system log (SYSLOG) with the VARY SYSLOG,HARDCPY command and rerun the program.

Table 41. Return and Reason Codes for the WTL Macro (continued)

Return Code	Reason Code	Meaning and Action
04	1C	<p>Meaning: Environmental error. The message was not logged in the system log, as requested, because the WTL limit has been reached. The operation log was not active at the time, so the message was not logged there either.</p> <p>Action: To log the message in the system log, do the following:</p> <ul style="list-style-type: none"> • Retry the request when the storage shortage has been relieved. • Issue a CONTROL M,LOGLIM command to change the allocated number of WTL SYSLOG buffers. • Change the LOGLIM value on the INIT statement in the CONSOLxx member of SYS1.PARMLIB. This new value will take effect at initialization. <p>If the problem persists, issue the CONTROL M,LOGLIM command first, and change the LOGLIM value in CONSOLxx at your next IPL.</p> <p>To log the message in the operations log, start the operations log and rerun the program.</p>
04	20	<p>Meaning: Environment error. The message was not logged in the operations log, as requested, because of storage problems. The system log was not active.</p> <p>Action: To log the message in the operations log, retry the request. If the problem persists, contact the IBM Support Center, providing the return and reason codes.</p> <p>To log the message in the system log also, start the system log and rerun the program.</p>
04	24	<p>Meaning: Environment error. The message was not logged in the system log because the WTL limit has been reached, and was not logged in the operation log because of storage problems.</p> <p>Action: To log the message in the operations log, retry the request. If the problem persists, contact the IBM Support Center, providing the return and reason codes.</p> <p>To log the message in the system log also, start the system log and rerun the program.</p>

Example 1

Write a message to the system log.

```
WTL    'THIS IS THE STANDARD FORMAT FOR THE WTL MACRO'
```

Example 2

Write a message to the system log specifying a prefix to identify the system log record.

```
WTL    'QL THIS FORMAT OF THE WTL USES THE OPTION KEYWORD',OPTION=PREFIX
```

Example 3

Build a parameter list for a message to be written to the system log.

```
LOGMSG    WTL    'FUNCTION XXX COMPLETE',MF=L
```

WTL Macro

Example 4

Write a message constructed in the list form of WTL.

```
WTL MF=(E, LOGMSG)
```

WTL—List Form

The list form of the WTL macro is used to construct a control program parameter list. The message parameter must be provided in the list form of the macro. The OPTION keyword is not permitted on the list form of the WTL macro.

Syntax

The list form of the WTL macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
-------------	--

b	One or more blanks must precede WTL.
---	--------------------------------------

WTL

b	One or more blanks must follow WTL.
---	-------------------------------------

'msg'	<i>msg</i> : Up to 126 characters.
-------	------------------------------------

,MF=L

Parameters

The parameters are explained under the standard form of the WTL macro with the following exception:

,MF=L

Specifies the list form of the WTL macro.

WTL—Execute Form

The execute form of the WTL macro uses a remote control program parameter list. The parameter list can be generated by the list form of WTL. You cannot modify the message in the execute form.

Syntax

The execute form of the WTL macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
-------------	--

↳ One or more blanks must precede WTL.

WTL

↳ One or more blanks must follow WTL.

MF=(E,*list addr*) *list addr*: RX-type address, or register (1) or (2) - (12).

 ,OPTION=PREFIX **Default:** OPTION=NOPREFIX
 ,OPTION=NOPREFIX

Parameters

The parameters are explained under the standard form of the WTL macro with the following exception:

MF=(E,*list addr*)

 Specifies the execute form of the WTL macro.

list addr is the name of a storage area to contain the parameters.

WTL Macro

WTO — Write to Operator

Description

Use the WTO macro to write a message to one or more operator consoles. See the section on communication in *z/OS MVS Programming: Authorized Assembler Services Guide* for more information on using WTO. *z/OS MVS Programming: Assembler Services Reference ABE-HSP* also contains information on WTO, with the exception of the AREAID, MSGTYP, CONNECT, SYSNAME, JOBID, JOBNAME, LINKAGE, SYNCH, and WSPARM parameters.

Environment

If you code **LINKAGE=SVC**, the requirements for the caller are:

Minimum authorization:	One of the following, depending on the parameters you code: <ul style="list-style-type: none">• Problem state and any key.• If you code the AREAID, MSGTYP, CONNECT, SYSNAME, JOBID, and JOBNAME parameters, one of the following:<ul style="list-style-type: none">– Supervisor state with PSW key 0-7– APF-authorized
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31- or 64-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in the primary address space.

If you code **LINKAGE=BRANCH**, the requirements for the caller are:

Minimum authorization:	One of the following, depending on the parameters you code: <ul style="list-style-type: none">• Supervisor state and PSW key 0 - 7.• If you code the WSPARM parameter, supervisor state with PSW key 0.
Dispatchable unit mode:	Task or SRB.
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	The caller may hold locks, but is not required to hold any.
Control parameters:	Must be in the primary address space.

Programming Requirements

Be aware of the following when coding the WTO macro:

- You must clear register 0 except under the following circumstances:
 - You are using register 0 to pass a 1-byte console ID with MCSFLAG=REG0. However, IBM recommends using the CONSID parameter rather than register 0.

WTO Macro

- You are using register 0 to pass a message identifier to connect multiple-line messages. However, IBM recommends using the CONNECT parameter rather than register 0.
- If the caller is disabled or issues WTO with the WSPARM parameter, the WTO and LOADWAIT parameter lists must be in fixed storage.
- When using any parameter with an address, the data being referenced must be accessible by the caller issuing the WTO.
- If the list and execute forms of the WTO macro are in separate modules, both modules must be assembled or compiled with the same level of WTO.
- When you're coding a reentrant program, make sure the WTO parameter list is generated correctly. To ensure this, you must code the same parameters on both forms, only when you code one or more of the following parameters:
 - TEXT=(*text addr*)
 - CONNECT
 - CONSID
 - CONSNAME
 - SYSNAME
 - CART
 - KEY
 - TOKEN
 - JOBNAME
 - JOBID
 - LINKAGE
 - WSPARM

On the list form, code only the parameter and the equal sign; do not code a parameter value as well. For example:

```
WTO 'text',CONSID=,MF=L
```

If you specify parameter values on the list form, the system issues an MNOTE and ignores the data.

- For any WTO parameters that allow a register specification, the value must be right-justified in the register.
- If WSPARM is not equal to zero, the wait state is loaded whether or not the message could be displayed or queued for hardcopy.
- If you specify the TEXT keyword for a multi-line WTO on the list form, you must omit *text addr* for each line, but include *line type*. If you specify *text addr*, the system ignores the data and issues an MNOTE. On the execute form, omit *line type* for each line, but include *text addr*.

Restrictions

- If an SRB-mode caller issues WTO without the JOBID or JOBNAME parameter, the WTO message will not have a job ID or jobname associated with it.
- You should issue a synchronous message only in serious system emergencies.
- There are two ways for authorized callers (supervisor state with PSW key 0-7) to issue multiple-line messages:
 1. You can issue a multiple-line WTO message of up to 255 lines with one WTO macro. If you are coding more than one multiple-line message, and you want to connect the messages, you must ensure that the left-most three bytes of register 0 are set correctly:
 - For the first request (of up to 255 lines), these three bytes must be zero.

- For subsequent requests, register 0 can contain the message identifier that the WTO service routine returns in register 1 after the first request. Note that you must left-justify the 8-bit identifier when you load it into register 0.
 - A WTO message with ROUTCDE=11 is sent to the JES2 system message dataset (SYSMSG) unless LINKAGE=BRANCH is specified. In this case, the message is not sent. If you want the WTO to go to SYSMSG, use LINKAGE=SVC with ROUTCDE=11. Whether you issue LINKAGE=BRANCH or LINKAGE=SVC with ROUTCDE=11, the message appears in the JES2 joblog.
2. The CONNECT parameter provides a way to connect multiple-line WTO messages. Therefore, an authorized caller actually can issue connect messages that total more than 255 lines.
- The caller cannot have an EUT FRR established.
 - When using the LINKAGE=BRANCH parameter, the system does not automatically delete a WTO issued by a caller in SRB mode. A caller in SRB mode must issue the DOM macro to explicitly delete any action message when the calling program ends.
 - If you specify LINKAGE=BRANCH, WTO ignores any data in register 0. Therefore, if you are connecting lines of a multi-line WTO, and you specified LINKAGE=BRANCH, you cannot put the CONNECT ID in register 0. You must use the CONNECT keyword.

Input Register Information

Before issuing the WTO macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output Register Information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register	Contents
0	Used as a work register by the system, unless WTO returns code X'20' in register 15. In that case, register 0 contains the number of active WTO buffers for the issuer's address space.
1	Message identification number if the macro completed normally. You can use this number to delete the message when it is no longer needed. If you are using the CONNECT parameter to connect WTO messages, store this value in the 4-byte CONNECT field and set register 1 to zero before issuing the next WTO. Otherwise, register 1 is used as a work register by the system.
2-13	Unchanged.
14	Used as a work register by the system.
15	Return code.

When control returns to the caller, the access registers (ARs) contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

WTO Macro

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance Implications

Users who cannot wait because of a WTO buffer shortage should use the MCSFLAG=BUSYEXIT parameter and then take appropriate action on the busy return.

SYNCH=YES causes the calling program to display the message, become disabled, and receive the reply synchronously.

Syntax

The standard form of the WTO macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
<i>b</i>	One or more blanks must precede WTO.
WTO	
<i>b</i>	One or more blanks must follow WTO.

<i>'msg'</i>	<i>msg</i> : Up to 126 characters.
<i>('text')</i>	<i>text</i> : Up to 126 characters.
<i>('text',line type)</i>	<i>text addr</i> : RX-type address or register (2) - (12).
<i>('text',line type)</i>	Note: If you code <i>'msg'</i> or <i>('text'...)</i> , it must be the first parameter you code.
<i>('text',line type)</i>	
TEXT=(<i>text addr</i>)	
TEXT=(<i>text addr,line type</i>)	
TEXT=((<i>text addr,line type</i>),...(<i>text addr,line type</i>))	

The permissible *line types*, text lengths, and maximum numbers of each line type are shown below:

line type	text	maximum number
C	35 char	1 C type
L	71 char	2 L type
D	71 char	255 D type
DE	71 char	1 DE type
	or	
E	None	1 E type

The maximum total number of lines that can be coded in one instruction is 255.

,ROUTCDE=(<i>routing code</i>)	<i>routing code</i> : Decimal digit from 1 to 128. The <i>routing code</i> is one or more codes, separated by commas, or a hyphen to indicate a range.
----------------------------------	--

,DESC=(<i>descriptor code</i>)	<i>descriptor code</i> : Decimal digit from 1 to 13. The <i>descriptor code</i> is one or more codes, separated by commas.
,AREAIID= <i>id char</i>	<i>id char</i> : Alphabetic character A - J, Z.
,MSGTYP=(<i>msg type</i>)	<i>msg type</i> : Any of the following N SESS,JOBNAMES Y SESS,STATUS SESS JOBNAMES,STATUS JOBNAMES SESS,JOBNAMES,STATUS STATUS
,MCSFLAG=(<i>flag name</i>)	<i>flag name</i> : Any combination of the following, separated by commas: REG0 HRDCPY RESP QREG0 REPLY NOTIME BRDCST NOCPY CMD BUSYEXIT
,CONNECT= <i>connect field</i>	<i>connect field</i> : RX-type address or register (2) - (12). Note: CONNECT is mutually exclusive with the CONSID, CONSNAME, SYSNAME, and SYNCH=YES parameters.
,CONSID= <i>console id</i>	<i>console id</i> : RX-type address or register (2) - (12).
,CONSNAME= <i>console name</i>	<i>console name</i> : RX-type address or register (2) - (12).
,SYSNAME= <i>system name</i>	<i>system name</i> : RX-type address or register (2) - (12).
,CART= <i>cmd/resp token</i>	<i>cmd/resp token</i> : RX-type address or register (2) - (12).
,KEY= <i>key</i>	<i>key</i> : RX-type address or register (2) - (12).
,TOKEN= <i>token</i>	<i>token</i> : RX-type address or register (2) - (12).
,JOBID= <i>job id</i>	<i>job id</i> : RX-type address or register (2) - (12).
,JOBNAME= <i>jobname</i>	<i>jobname</i> : RX-type address or register (2) - (12).
,LINKAGE=SVC ,LINKAGE=BRANCH	Default: SVC
,SYNCH=NO ,SYNCH=YES	Default: NO
,WSPARM=0 ,WSPARM= <i>wait state addr</i>	Default: 0 <i>wait state addr</i> : RX-type address or register (2) - (12).

Parameters

The parameters are explained as follows:

'msg'
('text')
('text',line type)

`('text',line type)`

`('text',line type)`

`TEXT=(text addr)`

`TEXT=(text addr,line type)`

`TEXT=((text addr,line type),...(text addr,line type))`

Specifies the message or multiple-line message to be written to one or more operator consoles.

The parameter `'msg'` is used to write a single-line message to the operator. In the format, the message must be enclosed in apostrophes, which do not appear on the console. It can include any character that can be used in a character (C-type) DC instruction.

To have apostrophes appear in the message text, use two apostrophes to get one to appear. For example, "Message Off" would appear on a display as 'Message Off'. When a program issues a WTO macro, the system translates the text; only standard printable EBCDIC characters are passed to MCS-managed display devices. The EBCDIC characters that can be displayed are listed in *z/OS MVS Programming: Assembler Services Guide*. All other characters are replaced by blanks. Unless the console has dual-case capability, lowercase characters are displayed or printed as uppercase characters.

The message is assembled as a variable-length record. The parameters `TEXT=(text addr)` and `TEXT=(text addr,line type)` represent a 4-byte address of a message to be displayed that consists of a 2-byte message length followed by the message text. The 2-byte message length describes the length of the message text only. There are no boundary requirements.

The parameters `('text')` and `(text addr,line type)` are used to write a multiple-line message to the operator. For a problem-state program, the message can be up to ten lines long; the system truncates the message at the end of the tenth line. The ten-line limit does not include the control line (message IEE932I), as explained under line type C below. The message can be up to 255 lines maximum.

Notes:

1. If the parameter `('text')` is coded without repetition, the message appears as a single-line message.
2. Specify all lines of a multiple-line WTO consistently with the message text or the TEXT keyword.
3. When coding the TEXT keyword for a multiple-line message:
 - Do not exceed the 70-character limit for the macro parameter value.
 - You can use the CONNECT parameter to connect subsequent lines of a multiple-line message if you cannot fit them into one macro invocation.
4. For a multiple-line message, you must clear the three high-order bytes of register 0.

The line type defines the type of information contained in the 'text' field of each line of the message:

- C** Indicates that the 'text' parameter is the text to be contained in the control line of the message. The control line normally contains a message title. C may only be coded for the first line of a multiple-line message. If this parameter is omitted and descriptor code 9 is coded, the system generates a control line (message IEE932I) containing only a message identification number. The control line remains static while you scroll through all the lines of a multi-line message displayed on an

MCS console (provided that the message is displayed in an out-of-line display area). Control lines are optional.

- L** Indicates that the 'text' parameter is a label line. Label lines contain message heading information; they remain static while you scroll through all the lines of a multi-line message displayed on an MCS console (provided that the message is displayed in an out-of-line display area). Label lines are optional. If coded, lines must either immediately follow the control line or another label line, or be the first line of the multiple-line message if there is no control line. Only two label lines may be coded per message. See "Embedding Label Lines in a Multi-line Message" in *z/OS MVS Programming: Authorized Assembler Services Guide* for additional information about how to include multiple label lines within a message.
- D** Indicates that the 'text' parameter contains the information to be conveyed to the operator by the multiple-line message. The data lines are paged while you scroll through all the lines of a multi-line message displayed on an MCS console.
- DE** Indicates that the 'text' parameter contains the last line of information to be passed to the operator. Specify DE on the last line of text of the WTO. If there is no text on the last line, specify E.
- E** Indicates that the previous line of text was the last line of text to be passed to the operator. The 'text' parameter, if any, coded with a line type of E is ignored. Specify E on the last line of the WTO if that line has no text. If the last line has text, specify DE.

,ROUTCDE=(routing code)

Specifies the routing code or codes to be assigned to the message.

The routing codes are:

Message Routing Code	Definition
1	Master console action
2	Master console information
3	Tape pool
4	Direct access pool
5	Tape library
6	Disk library
7	Unit record pool
8	Teleprocessing control
9	System security
10	System error/maintenance/system programmer information
11	Programmer information
12	Emulators
13-20	Reserved for customer use
21-28	Reserved for IBM- or customer-defined subsystem use
29-41	Reserved for IBM
42	General information about JES2 or JES3
43-64	Reserved for JES2 or JES3
65-96	Messages associated with particular processors
97-128	Messages associated with particular devices

WTO Macro

If you omit the ROUTCDE, DESC, and CONSID or CONSNAME keywords, the system uses the routing code specified on the ROUTCODE keyword on the DEFAULT statement in the CONSOLxx member of SYS1.PARMLIB.

Note: Routing codes 1, 2, 3, 4, 7, 8, 10, and 42 cause hard copy of the message when display consoles are used, or more than one console is active. All other routing codes may go to hard copy as a PARMLIB option or as a result of a VARY HARDCPY command.

,DESC=(descriptor code)

Specifies the message descriptor code or codes to be assigned to the message. Descriptor codes 1 through 6, 11 and descriptor code 12 are mutually exclusive. Codes 7 through 10, and 13, can be assigned in combination with any other code.

The descriptor codes are:

Message Descriptor Code	Definition
1	System failure
2	Immediate action required
3	Eventual action required
4	System status
5	Immediate command response
6	Job status
7	Delete message when job step terminates
8	Out-of-line message
9	Operator request
10	Dynamic status displays
11	Critical eventual action requested
12	Important information messages
13	Message previously automated

Action messages many have an * sign or @ sign displayed before the first character of the message. The * sign indicates that the WTO was issued by an authorized program. The @ sign indicates that the WTO was issued by an unauthorized program.

All WTO messages with descriptor codes 1, 2, or 11 are action messages that have an asterisk (*) sign displayed before the first character of the message. This indicates a need for operator action. These action messages will cause the audible alarm to sound on operator consoles so-equipped. On operator consoles that support color, descriptor codes determine the color in which a message should be displayed. Colors can indicate the type of action you need to take depending on your installation setup. The colors used for different descriptor codes are described in *z/OS MVS System Commands*.

The system holds messages with descriptor codes 1, 2, 3, or 11 until you delete them. When you no longer need messages with descriptor codes 1, 2, 3, or 11, you should delete those messages using the DOM macro. If messages with descriptor codes 1, 2, 3, or 11 also have descriptor code 7, the system deletes them automatically at task termination. The system adds descriptor code 7 to all messages with descriptor code 1 or 2.

If descriptor code 7 is specified, the system deletes the message automatically when the job step that issued it ends.

The message processing facility (MPF) can suppress messages. For MPF to suppress messages, the hardcopy log must be active. The suppressed messages do not appear on any console; they do appear on the hardcopy log.

,AREAID=*id char*

Specifies a display area of the console screen on which a multiple-line message is to be written.

Valid area IDs are A through J and Z. A through J refer to out-of-line areas defined on an MCS console by the CONTROL A command. The character Z designates the message area (the screen's general message area, rather than a defined display area); it is assumed if nothing is specified. The areas are explained in *z/OS MVS System Commands*.

Notes:

1. WTO ignores this keyword on single-line invocations.
2. When you specify AREAID, you must specify descriptor codes 8 and 9.
3. If you specify this parameter, the area could be overlaid by a currently running dynamic display. Support for queuing messages with descriptor code 8 is by console ID only. You must specify a console explicitly using the CONSID or CONSNAME parameters on WTO.
4. You can use the CONVCON macro to validate an area ID.

,MSGTYP=(*msg type*)

Specifies how the message is to be routed to consoles on which the MONITOR command is active. If you specify anything other than MSGTYP=N, which is the default, your message will be routed according to your specification on MSGTYP, and the ROUTCDE parameter is ignored.

For SESS, JOBNAMES, or STATUS, the message is to be routed to the console that issued the MONITOR SESS, MONITOR JOBNAMES, or MONITOR STATUS command, respectively. When the message type is identified by the operating system, the message is routed to those consoles that requested the information.

For Y, the message type describes what functions (MONITOR SESS, MONITOR JOBNAMES, and MONITOR STATUS) are desired. N, or omission of the MSGTYP parameter, indicates that the message is to be routed as specified in the ROUTCDE parameter. Y creates an area in the WTO parameter list in which you can set message type information if you are coding a WTO without any of the following parameters:

- KEY
- TOKEN
- CONSID
- CONSNAME
- TEXT
- REPLYISUR
- CART
- LINKAGE
- SYNCH

IBM recommends that you do not use MSGTYP=Y.

,MCSFLAG=(*flag name*)

Specifies one or more flags whose meanings are shown below:

Table 42. MCSFLAG Flag Names

Flag Name	Meaning
REG0	Queue the message to the console whose console ID is passed in register 0. You can obtain valid console IDs by issuing the DISPLAY CONSOLE command. You can use register 0 to pass a 1-byte console ID (right-justified and padded to the left with zeros) to identify the console to receive the message. However, IBM recommends you use the CONSID parameter instead of register 0.
RESP	The WTO is an immediate command response.
REPLY	This WTO is a reply to a WTOR.
BRDCST	Broadcast the message to all active consoles.
HRDCPY	Queue the message for hard copy only.
QREG0	Queue the message to the console whose console ID is passed in register 0. You can use register 0 to pass a 1-byte console ID (right-justified and padded to the left with zeros) to identify the console to receive the message. However, IBM recommends you use the CONSID parameter instead of register 0.
NOTIME	Do not append time to the message.
NOCPY	Do not queue the message for hard copy.
CMD	The WTO is a recording of a system command issued for hardcopy log purposes.
BUSYEXIT	If there are no message or console buffers for either MCS or JES3, or there is a JES3 staging area shortage, the WTO is terminated with a X'20' return code and a reason code (in register 0) equal to the number of active WTO buffers for the issuer's address space. If you do not specify BUSYEXIT, most users of WTO will wait until WTO buffers are available. Specify BUSYEXIT if your task cannot tolerate a wait for WTO buffers.

Notes:

1. Do not use REG0 or QREG0 if you use the CONSID or CONSNAME parameters.
2. MCSFLAG=HRDCPY and SYNCH=YES are mutually exclusive.

,CONNECT=connect field

Specifies a field containing the 4-byte message ID of the previous WTO to which this WTO is to be connected. This message ID is obtained as an output parameter (returned in register 1) from the previous WTO. If a register is used, it contains the address of the message ID.

CONNECT is valid only for continuation of multiple-line messages. When you specify this parameter in the list form, code it as CONNECT= with nothing after the equal sign.

This parameter is mutually exclusive with the CONSID, CONSNAME, and SYSNAME parameters.

Note: If you specify LINKAGE=SVC, you can still use register 0, as mentioned at the beginning of the WTO macro description, to connect WTO messages. If you specify both register 0 and CONNECT, however, the system uses the CONNECT parameter. IBM recommends that you use the CONNECT parameter.

,CONSID=console id

Specifies a 4-byte field containing the ID of the console to receive a message.

Use this ID in place of a console ID in register 0. If you specify a 4-byte console ID, or if you specify a console ID for an extended MCS console, you must use CONSID instead of register 0. If you specify a 1-byte console ID, you must right-justify it and pad to the left with zeroes. Issue the DISPLAY CONSOLE command for a list of valid console IDs.

Notes:

1. If you code the CONSID parameter using a register, the register must contain the console ID itself, rather than the address of the console ID.
2. When you code CONSID on the list form of WTO, code it as CONSID= with nothing after the equal sign.
3. Do not use both CONSID and register 0 to pass a console ID, because the results are unpredictable. Be sure to clear the low-order byte of register 0 if you add the CONSID parameter to an existing invocation of WTO.
4. CONSID is mutually exclusive with the CONNECT, CONSNAME, and SYSNAME parameters.

,CONSNAME=*console name*

Specifies an 8-byte field containing a 2- through 8-character name, left-justified and padded with blanks, of the console to receive a message. When you specify this parameter in the list form, code it as CONSNAME= with nothing after the equal sign.

Do not use CONSNAME to pass a console name, together with register 0 to pass a console ID, because the results are unpredictable. Be sure to clear the low-order byte of register 0 if you add the CONSNAME parameter to an existing invocation of WTO.

This parameter is mutually exclusive with the CONSID, CONNECT, and SYSNAME parameters.

,SYSNAME=*system name*

Specifies an 8-byte input field containing a system name to be associated with this message. The system name is that of the system from which the caller issues the WTO. When you specify this parameter in the list form, code it as SYSNAME= with nothing after the equal sign.

This parameter is mutually exclusive with the CONNECT parameter.

,CART=*cmd/resp token*

Specifies an 8-character input field containing a command and response token to be associated with this message. The command and response token is used to associate user information with a command and its command response. You can supply any value as a command and response token. When you specify this parameter in the list form, code it as CART= with nothing after the equal sign.

,KEY=*key*

Specifies an input field containing an 8-byte key to be associated with this message. The key must be EBCDIC if used with the MVS DISPLAY R command for retrieval purposes, but it must not be '*'. The key must be left-justified, and padded on the right with blanks. If a register is used, it contains the address of the key. When you specify this parameter in the list form, code it as KEY= with nothing after the equal sign.

,TOKEN=*token*

Specifies an input field containing a 4-byte token to be associated with this message. This field is used to identify a group of messages that can be deleted by a DOM macro that includes TOKEN. The token must be unique within an

address space and can be any value. When you specify this parameter in the list form, code it as `TOKEN=` with nothing after the equal sign.

Note: When you code the `TOKEN` parameter using a register, the register must contain the token itself, rather than the address of the token.

,JOBID=*job id*

Specifies an 8-byte input field containing an ID that identifies the issuer of the WTO message. When you specify this parameter in the list form, code it as `JOBID=` with nothing after the equal sign.

,JOBNAME=*jobname*

Specifies an 8-byte input field containing a name that identifies the issuer of the WTO message. When you specify this parameter in the list form, code it as `JOBNAME=` with nothing after the equal sign.

,LINKAGE=SVC

,LINKAGE=BRANCH

Specifies how control is to pass to the WTO service.

`LINKAGE=SVC` indicates that the linkage is by a supervisor call. If `LINKAGE` is not specified, this is the default.

`LINKAGE=BRANCH` indicates that the linkage is by a branch-and-link. This parameter is used by programs that run at times when an `SVC` cannot be issued, and by programs that require the WTO request to be handled synchronously.

When you specify this parameter in the list form, code it as `LINKAGE=` with nothing after the equal sign.

If you specify `LINKAGE=BRANCH`, you cannot put the `CONNECT` value in register 0. You must use the `CONNECT` keyword.

,SYNCH=NO

,SYNCH=YES

Specifies whether the WTO request processes synchronously with the caller.

`SYNCH=NO`, the default, indicates that the request is not processed synchronously.

`SYNCH=YES` indicates that the request is to be processed synchronously. This parameter is used in error and recovery environments, when normal message processing cannot be used. The message is sent to a console, where it is held on the screen for up to ten seconds, before control is returned to the caller. A copy of the message is queued for transcription to the hardcopy log.

If you specify `SYNCH=YES`:

- You must specify the parameter `LINKAGE=BRANCH`.
- The message text must be 14 lines or less.
- The following parameters are mutually exclusive: `CONNECT`, `AREAID`, and `MCSFLAG=HRDCPY`.

Your installation can determine which consoles can receive synchronous messages by using the `SYNCHDEST` parameter in the `CONSOLxx` member of `SYS1.PARMLIB`. For additional information on the `SYNCHDEST` parameter, see *z/OS MVS Initialization and Tuning Reference*.

,WSPARM=0

,WSPARM=*wait state addr*

Specifies whether a wait state is associated with this message.

A value of zero indicates that there is no wait state associated with this message. If you do not specify WSPARM, this is the default.

A nonzero value indicates either the address of a LOADWAIT parameter list or a register containing a pointer to the parameter list. The LOADWAIT macro generates the LOADWAIT parameter list. When you specify this parameter in the list form, code it as WSPARM= with nothing after the equal sign.

This parameter requires the SYNCH=YES and LINKAGE=BRANCH parameters.

ABEND Codes

WTO might abnormally terminate with abend code X'D23'. See *z/OS MVS System Codes* for an explanation and programmer response for this code.

Return and Reason Codes

When the WTO macro returns control to your program, GPR 15 contains one of the following hexadecimal return codes.

Table 43. Return Codes for the WTO Macro

Return Code	Meaning and Action
00	<p>Meaning: Processing completed successfully.</p> <p>Action: None.</p>
04	<p>Meaning: Program error. One of the following occurred:</p> <ul style="list-style-type: none"> The number of lines passed was 0. The request was ignored. The number of lines passed was greater than 255 and you did not specify the CONNECT parameter. Only 255 lines were processed. The message text length for a line was less than 1. All lines up to the error line were processed. <p>Action: Do one or more of the following:</p> <ul style="list-style-type: none"> Make sure your text is properly referenced. If you are using the TEXT parameter, make sure it is pointing to valid data. Use the CONNECT parameter to organize your message into groups of 255 or fewer lines of text. Make sure your message text is defined correctly. If you are using the TEXT parameter, make sure the first two bytes of data in the area pointed to by the TEXT parameter value contain the length of the message text. <p>In all cases, correct the problem and retry the request.</p>
08	<p>Meaning: Program error. The connecting message ID (passed in register 0 or as specified by the CONNECT parameter value) does not match any on the queue. The request was ignored.</p> <p>Action: Verify the connect ID value, correct the problem, and retry the request.</p>
0C	<p>Meaning: Program error. A line type is not valid. An end was forced at the point of the error unless the first line is an E line, in which case the request was ignored. All messages up to this one in the multi-line request were processed.</p> <p>Action: Determine if a line type value on your multi-line message was not syntactically correct. Correct the problem and retry the request.</p>
10	<p>Meaning: Environmental error. Multiple line WTO processing has been terminated after 1000 lines during a WTO buffer shortage.</p> <p>Action: Retry the request when the buffer storage constraint has been relieved.</p>
20	<p>Meaning: Environmental error. WTO processing has been terminated because it would have caused a wait state, and BUSYEXIT was specified. Register 0 contains the number of active WTO buffers for the issuer's address space.</p> <p>Action: Retry the request when the buffer storage constraint has been relieved.</p>

Table 43. Return Codes for the WTO Macro (continued)

Return Code	Meaning and Action
24	<p>Meaning: Environmental error. BUSYEXIT was not specified and WTO processing has been terminated. Processing the WTO might have caused a WTO buffer shortage that would require a relPL. If this WTO was part of a multiple line WTO, the multiple line WTO is ended.</p> <p>Action: Retry the request when the buffer storage constraint has been relieved.</p>
30	<p>Meaning: Environmental error. For routing code 11, the required resource was not available and the request was ignored. For any other routing code, the request was processed.</p> <p>Action: Retry the request when the resource you need is available.</p>
40	<p>Meaning: Environmental error. WTO was issued with LINKAGE=BRANCH; insufficient storage was available to queue the message for delayed issue. If SYNCH=NO was specified, the message was not queued for delayed issue. If SYNCH=YES was specified, the message was delivered for display, but not queued for hardcopy.</p> <p>Action: If you want the message to be delivered to the destination you requested, reissue the request. If the message was an action message that was not displayed, a DOM request is not required.</p>
44	<p>Meaning: Environmental error. WTO was issued with LINKAGE=BRANCH, SYNCH=YES; no usable console was available. The message was queued for hardcopy, but not delivered for display.</p> <p>Action: If you want the message to be delivered to the destination you requested, reissue the request. If the message was an action message that was not displayed, a DOM request is not required.</p>
48	<p>Meaning: Environmental error. WTO was issued with LINKAGE=BRANCH, SYNCH=YES; no usable console was available and insufficient storage was available to queue the message for delayed issue. The message was not delivered for display, nor queued for hardcopy.</p> <p>Action: If you want the message to be delivered to the destination you requested, reissue the request. If the message was an action message that was not displayed, a DOM request is not required.</p>
4C	<p>Meaning: Environmental error. WTO was issued with LINKAGE=BRANCH; no storage was available for the use of WTO processing. If WSPARM=0, no processing was done. If WSPARM does not equal zero, WTO loaded the wait state, but performed no other processing.</p> <p>Action: If you want the message to be delivered to the destination you requested, reissue the request. If the message was an action message that was not displayed, a DOM request is not required.</p>

Example 1

Issue a WTO with routing codes 1 and 10, descriptor code 2, using the BUSYEXIT option of the MCSFLAG parameter. Send an immediate action message to the master console. If there is a WTO buffer shortage, WTO will return rather than wait for an available buffer.

```
WTO      'USR001I CRITICAL RESOURCE SHORTAGE DETECTED',X
         ROUTCDE=(1,10),                               X
         DESC=(2),MCSFLAG=BUSYEXIT
```

Example 2

Issue an important information message to a console whose name is defined in field MYCONS. This message is issued using the branch-entry option.

```

WTO      'USR123I DO NOT SUBMIT JOBS REQUESTING THREE OR MORE TAPX
        ES DURING FIRST SHIFT',
        DESC=(12),CONSNAME=MYCONS,LINKAGE=BRANCH,SYNCH=NO
        .
        .
        .
MYCONS DC   CL8'SCHDCONS'

```

Example 3

Issue a multi-line message using the TEXT parameter. This is an important information message which does not have a time stamp and is not sent to the hardcopy log.

```

WTO      TEXT=((REMIND1,D),(REMIND2,D),(REMIND3,DE)),X
        DESC=(12),MCSFLAG=(NOTIME,NOCPY)
        .
        .
        .
REMIND1 DC AL2(L'RMD1TXT)
RMD1TXT DC C'USR003I REMINDER:  THE SYSTEM IS NOT AVAILABLE FOR USEX
        ON THIRD SHIFT'
REMIND2 DC AL2(L'RMD2TXT)
RMD2TXT DC C'FOR SPECIAL REQUESTS CONTACT SYSTEM SUPPORT'
REMIND3 DC AL2(L'RMD3TXT)
RMD3TXT DC C'CALL THE STATUS DESK FOR FURTHER INFORMATION'

```

WTO—List Form

Use the list form of the WTO macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to store the parameters.

Syntax

The list form of the WTO macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
-------------	--

b	One or more blanks must precede WTO.
---	--------------------------------------

WTO

b	One or more blanks must follow WTO.
---	-------------------------------------

<i>'msg'</i>	<i>msg</i> : Up to 126 characters.
--------------	------------------------------------

<i>('text')</i>	<i>text</i> : Up to 126 characters.
-----------------	-------------------------------------

<i>('text',line type)</i>	
---------------------------	--

Notes:

1. If you code *'msg'* or *('text'...)*, it must be the first parameter you code.
2. For a single-line WTO, the parameter value is not required on TEXT for the list form. Code only TEXT=. Then code TEXT=(*text addr*) on the execute form.

TEXT=

TEXT=((*line type*),(*line type*),...(*line type*))

WTO Macro

The permissible *line types*, text lengths, and maximum numbers of each line type are shown below:

line type	text	maximum number
C	35 char	1 C type
L	71 char	2 L type
D	71 char	255 D type
DE	71 char	1 DE type
	or	
E	None	1 E type

The maximum total number of lines that can be coded in one instruction is 255.

,ROUTCDE=(<i>routing code</i>)	<i>routing code</i> : Decimal digit from 1 to 128. The <i>routing code</i> is one or more codes, separated by commas, or a hyphen to indicate a range.
,DESC=(<i>descriptor code</i>)	<i>descriptor code</i> : Decimal digit from 1 to 13. The <i>desc code</i> is one or more codes, separated by commas.
,AREAID= <i>id char</i>	<i>id char</i> : Alphabetic character A - J, Z.
,MSGTYP=(<i>msg type</i>)	<i>msg type</i> : Any of the following N SESS,JOBNAMES Y SESS,STATUS SESS JOBNAMES,STATUS JOBNAMES SESS,JOBNAMES,STATUS STATUS
,MCSFLAG=(<i>flag name</i>)	<i>flag name</i> : Any combination of the following, separated by commas: REG0 HRDCPY RESP QREG0 REPLY NOTIME BRDCST NOCPY CMD BUSYEXIT
,CONNECT=	Parameter value not required for list form. Code only,CONNECT=. If you code CONNECT on the execute form of WTO, you must code the same parameter on the list form.
,CONSID=	Parameter value not required for list form. Code only,CONSID= (or ,CONSNAME=). If you code CONSID (or CONSNAME) on the execute form of WTO, you must code the same parameter on the list form.
,CONSNAME=	
,SYSNAME=	Parameter value not required for list form. Code only,SYSNAME=. If you code SYSNAME on the execute form of WTO, you must code the same parameter on the list form.
,CART=	Parameter value not required for list form. Code only,CART=. If you code CART on the list form of WTO, you must code CART on the execute form.
,KEY=	Parameter value not required for list form. Code only,KEY=. If you code KEY on the list form of WTO, you must code KEY on the execute form.
,TOKEN=	Parameter value not required for list form. Code only,TOKEN=. If you code TOKEN on the list form of WTO, you must code TOKEN on the execute form.

,JOBID=	Parameter value not required for list form. Code only,JOBID=. If you code JOBID on the list form of WTO, you must code JOBID on the execute form.
,JOBNAME=	Parameter value not required for list form. Code only,JOBNAME=. If you code JOBNAME on the list form of WTO, you must code JOBNAME on the execute form.
,LINKAGE=	Parameter value not required for list form. Code only,LINKAGE=. If you code LINKAGE on the list form of WTO, you must code LINKAGE on the execute form.
,SYNCH=NO ,SYNCH=YES	Default: NO
,WSPARM=	Parameter value not required for list form. Code only,WSPARM=. If you code WSPARM on the list form of WTO, you must code WSPARM on the execute form.
,MF=L	

Parameters

The parameters are explained under the standard form of the WTO macro with the following exception:

,MF=L

Specifies the list form of the WTO macro.

WTO—Execute Form

Use the execute form of the WTO macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

The message cannot be modified on the execute form of the macro if you code inline text (*'msg'* or (*'text'...*)) on the list form.

Syntax

The execute form of the WTO macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
␣	One or more blanks must precede WTO.
WTO	
␣	One or more blanks must follow WTO.

WTO Macro

TEXT=(*text addr*)

TEXT=((*text addr*),(*text addr*),...(*text addr*))

text addr: RX-type address or register (2) - (12).

Notes:

1. If you code TEXT=(*text addr*) on the execute form of WTO, you must code TEXT= on the list form.
2. If you specify inline text on the list form ('*msg*' or ('*text*'...)), do not code the TEXT keyword on the execute form.

,CONNECT=*connect field*

connect field: RX-type address or register (2) - (12). If you code CONNECT on the execute form of WTO, you must code the same parameter on the list form.

,CONSID=*console id*

,CONSNAME=*console name*

console id: RX-type address or register (2) - (12).

console name: RX-type address or register (2) - (12).

If you code CONSID (or CONSNAME) on the execute form of WTO, you must code the same parameter on the list form.

,SYSNAME=*system name*

system name: RX-type address or register (2) - (12). If you code SYSNAME on the execute form of WTO, you must code the same parameter on the list form.

,CART=*cmd/resp token*

cmd/resp token: RX-type address or register (2) - (12). If you code CART on the execute form of WTO, you must code CART on the list form.

,KEY=*key*

key: RX-type address or register (2) - (12). If you code KEY on the execute form of WTO, you must code KEY on the list form.

,TOKEN=*token*

token: RX-type address or register (2) - (12). If you code TOKEN on the execute form of WTO, you must code TOKEN on the list form.

,JOBID=*job id*

job id: RX-type address or register (2) - (12). If you code JOBID on the execute form of WTO, you must code JOBID on the list form.

,JOBNAME=*jobname*

jobname: RX-type address or register (2) - (12). If you code JOBNAME on the execute form of WTO, you must code JOBNAME on the list form.

,LINKAGE=SVC

,LINKAGE=BRANCH

Default: SVC

If you code LINKAGE on the execute form of WTO, you must code LINKAGE on the list form.

,SYNCH=NO

,SYNCH=YES

Default: NO

,WSPARM=0

,WSPARM=*wait state addr*

Default: 0

wait state addr: RX-type address or register (2) - (12).

If you code WSPARM on the execute form of WTO, you must code WSPARM on the list form.

,MF=(E,*list addr*)

list addr: RX-type address, or register (1) - (12).

Parameters

The parameters are explained under the standard form of the WTO macro, with the following exception:

,MF=(E, *list addr*)

Specifies the execute form of the WTO macro.

list addr specifies the area that the system uses to store the parameters.

Example

Write a message with a prebuilt parameter list pointed to by register 1.

```
WTO      MF=(E,(1))
```

WTO Macro

WTOR — Write to Operator with Reply

Description

The WTOR macro causes a message requiring a reply to be written to one or more operator consoles and the hardcopy log. The macro also provides the information required by the system to return the reply to the issuing program. See *z/OS MVS Programming: Authorized Assembler Services Guide* for more information on using the WTOR macro.

For information about how to select a macro for an MVS/SP version other than the current version, see “Compatibility of MVS Macros” on page 1.

The description of the WTOR macro follows. The WTOR macro is also described in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*, with the exception of the MSGTYP, SYSNAME, JOBID, JOBNAME, LINKAGE, and SYNCH parameters.

Environment

If you code **LINKAGE=SVC**, the requirements for the caller are:

Minimum authorization:	One of the following, depending on the parameters you code: <ul style="list-style-type: none">• Problem state and PSW key 0-7.• If you code the MSGTYP, SYSNAME, JOBID, JOBNAME, and SYNCH parameters, one of the following:<ul style="list-style-type: none">– Supervisor state with PSW key 0-7– APF-authorized
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31- or 64-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	No locks held
Control parameters:	Must be in the primary address space

If you code **LINKAGE=BRANCH**, the requirements for the caller are:

Minimum authorization:	One of the following: <ul style="list-style-type: none">• Supervisor state with PSW key 0-7• APF-authorized
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled or disabled for I/O and external interrupts.
Locks:	The caller may hold locks, but is not required to hold any.
Control parameters:	Must be in the primary address space.

Programming Requirements

Be aware of the following when coding the WTOR macro:

WTOR Macro

- The caller must clear register 0 unless using register 0 to pass a 1-byte console ID with MCSFLAG=REG0. However, IBM recommends using the CONSID parameter rather than register 0.
- The parameter list for WTOR must begin on a fullword boundary.
- If the caller is disabled, the WTOR parameter list, reply area, and reply ECB must be in fixed storage.
- If the list and execute forms of the WTOR macro are in separate modules, both modules must be assembled or compiled with the same level of WTOR.
- When you're coding a reentrant program, make sure the WTOR parameter list is generated correctly. To ensure this, you must code the same parameters on both forms, only when you code one or more of the following parameters:
 - RPLYISUR
 - CONSID
 - CONSNAME
 - SYSNAME
 - CART
 - KEY
 - TOKEN
 - JOBNAME
 - JOBID
 - LINKAGE

On the list form, code only the parameter and the equal sign; do not code a parameter value as well. If you specify parameter values on the list form, the system issues an MNOTE and ignores the data.

- For any WTOR keywords that allow a register specification, the value must be right-justified in the register.
- If you specify the TEXT keyword for a multi-line WTOR, you must code its parameters in the following way:
 - On the list form, omit *text addr* for each line, but include *line type*. If you specify *text addr*, the system ignores the data and issues an MNOTE.
 - On the execute form, omit *line type* for each line, but include *text addr*.

Restrictions

If the LINKAGE=BRANCH parameter is specified, the SYNCH=YES parameter is required.

You can issue a multi-line WTOR only if you specify LINKAGE=BRANCH, SYNCH=YES.

Issue a synchronous message only if your program is in a state in which it cannot issue an ordinary WTOR (LINKAGE=SVC), and you must receive operator input before continuing.

The caller cannot have an EUT FRR established.

Input Register Information

Before issuing the WTOR macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output Register Information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register	Contents
0	Used as a work register by the system
1	Message identification number if the macro completed normally (you can use this number to delete the message when it is no longer needed); otherwise, used as a work register by the system
2-13	Unchanged
14	Used as a work register by the system
15	Return code

When control returns to the caller, the access registers (ARs) contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance Implications

SYNCH=YES causes the calling program to display the message, become disabled, and receive the reply synchronously.

Syntax

The standard form of the WTOR macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
<i>b</i>	One or more blanks must precede WTOR.
WTOR	
<i>b</i>	One or more blanks must follow WTOR.

```
'msg',reply addr,reply length,ecb addr
('text',reply addr,reply length,ecb addr)
(('text',line type),...('text',line type)),reply addr,reply length,ecb addr
TEXT=(text addr,reply addr,reply length,ecb addr)
TEXT=((text addr,line type),reply addr,reply length,ecb addr)
TEXT=(((text addr,line type),... (text addr, line type)),reply addr,reply length,ecb addr)
```

Note: If you code 'msg'... or ('text'...), it must be the first parameter you code.
msg: Up to 122 characters.
text: Up to 122 characters.
text addr: RX-type address or register (2) - (12).
reply addr: A-type address, or register (2) - (12).
reply length: Symbol, decimal digit, or register (2) - (12). The minimum length is 1; the maximum length is 119.

WTOR Macro

ecb addr: A-type address, or register (2) - (12).

The permissible *line types*, text lengths, and maximum numbers are shown below:

line type	text	maximum number
C	31 char	1 C type
L	67 char	2 L type
D	67 char	14 D type
DE	67 char	1 DE type
	or	
E	None	1 E type

The maximum total of lines that can be coded in one instruction is 14.

,ROUTCDE=(*routing code*)

routing code: Decimal digit from 1 to 128. The *routing code* is one or more codes, separated by commas, or a hyphen to indicate a range.

,DESC=(*descriptor code*)

descriptor code: Decimal number 7 or 13. If you code both 7 and 13, separate them with commas.

,MSGTYP=(*msg type*)

msg type: Any of the following:

N	SESS,JOBNAMES
Y	SESS,STATUS
SESS	JOBNAMES,STATUS
JOBNAMES	SESS,JOBNAMES,STATUS
STATUS	

,MCSFLAG=(*flag name*)

flag name: Any combination of the following, separated by commas:

REG0	HRDCPY
RESP	QREG0
REPLY	NOTIME
BRDCST	NOCPY
CMD	

,RPLYISUR=*reply console*

reply console: RX-type address or register (2) - (12).

,CONSID=*console id*

console id: RX-type address or register (2) - (12).

,CONSNAME=*console name*

console name: RX-type address or register (2) - (12).

,SYSNAME=*system name*

system name: RX-type address or register (2) - (12).

,CART=*cmd/resp token*

cmd/resp token: RX-type address or register (2) - (12).

,KEY=*key*

key: RX-type address or register (2) - (12).

,TOKEN=*token*

token: RX-type address or register (2) - (12).

,JOBID=*job id*

job id: RX-type address or register (2) - (12).

,JOBNAME=*jobname*

jobname: RX-type address or register (2) - (12).

,LINKAGE=SVC

Default: SVC

,LINKAGE=BRANCH

,SYNCH=NO

Default: NO

,SYNCH=YES

Parameters

The parameters are explained as follows:

```
'msg',reply addr,reply length,ecb addr
('text',reply addr,reply length,ecb addr)
(('text',line type,...'text',line type),reply addr,reply length,ecb addr)
TEXT=(text addr,reply addr,reply length,ecb addr)
TEXT=((text addr,line type),reply addr,reply length,ecb addr)
TEXT=(((text addr,line type),... (text addr, line type)),reply addr,reply length,ecb
addr)
```

Specifies the message or multiple-line message to be written to one or more operator consoles.

Use the *'msg'* parameter to write a single-line message to the operator. Enclose the message in apostrophes. The apostrophes do not appear on the console. You can include any character that can be used in a character (C-type) DC instruction.

To have apostrophes appear in the message text, use two apostrophes to get one to appear. For example, "Message Off" would appear on a display as 'Message Off'. When a program issues a WTOR macro, the system translates the text; only standard printable EBCDIC characters are passed to MCS-managed display devices. The EBCDIC characters that can be displayed are listed in "Timing and Communication" in *z/OS MVS Programming: Assembler Services Guide*. All other characters are replaced by blanks. Unless the console has dual-case capability, lowercase characters are displayed or printed as uppercase characters.

The message is assembled as a variable-length record. *text addr* represents a 4-byte address of a message to be displayed that consists of a 2-byte message length followed by the message text. The 2-byte message length describes the length of the message text only. There are no boundary requirements.

Use the parameters *('text')* and *(text addr,line type)* to write a multiple-line message to the operator. For a problem-state program, the message can be up to ten lines long; the system truncates the message at the end of the tenth line. The ten-line limit does not include the control line (message IEE932I), as explained under line type C below. The message can be up to 14 lines maximum.

Notes:

1. You can issue a multi-line WTOR only if you specify SYNCH=YES. See the SYNCH parameter description for information about its use.
2. If you code the parameter *('text')* without repetition, the message appears as a single-line message.
3. Specify all lines of a multiple-line WTOR consistently with the message text or the TEXT keyword. When coding the TEXT keyword for a multiple-line message, do not exceed the 67-character limit for the macro parameter value.

The line type defines the type of information contained in the 'text' field of each line of the message:

- C** Indicates that the 'text' parameter is the text to be contained in the control line of the message. The control line normally contains a message title. C may be coded only for the first line of a multiple-line

message. The control line remains static while you scroll through all the lines of a multi-line message displayed on an MCS console (provided that the message is displayed in an out-of-line display area). Control lines are optional.

- L** Indicates that the 'text' parameter is a label line. Label lines contain message heading information; they remain static while you scroll through all the lines of a multi-line message displayed on an MCS console (provided that the message is displayed in an out-of-line display area). Label lines are optional. If coded, lines must either immediately follow the control line or another label line or be the first line of the multiple-line message if there is no control line. Only two label lines may be coded per message. See "Embedding Label Lines in a Multi-line Message" in *z/OS MVS Programming: Authorized Assembler Services Guide* for additional information about how to include multiple label lines within a message.
- D** Indicates that the 'text' parameter contains the information to be conveyed to the operator by the multiple-line message. The data lines are paged during framing operations on a display console.
- DE** Indicates that the 'text' parameter contains the last line of information to be passed to the operator. Specify DE on the last line of text of the WTO. If there is no text on the last line, specify E.
- E** Indicates that the previous line of text was the last line of text to be passed to the operator. The 'text' parameter, if any, coded with a line type of E is ignored. Specify E on the last line of the WTO if that line has no text. If the last line has text, specify DE.

Note: All WTO messages are action messages. An indicator appears before the first character of an action message to indicate a need for operator action. Action messages will cause the audible alarm to sound on operator consoles so-equipped.

reply addr specifies the address in virtual storage of the area into which the system is to place the reply. The reply is left-justified at this address.

reply length specifies the length, in bytes, of the reply message.

ecb addr specifies the address of the event control block (ECB) to be used by the system to indicate the completion of the reply and the ID of the replying console. The ECB address must point to a fullword boundary. After the system receives the reply, the ECB appears as follows:

Offset	Length(bytes)	Contents
0	1	Completion code
1	2	Not an intended programming interface
3	1	Console ID in hexadecimal

Note: Use RPLYISUR to obtain the 4-byte console id and console name of the console issuing the reply.

,ROUTCDE=(*routing code*)

Specifies the routing code or codes to be assigned to the message.

The routing codes are:

Message Routing Code	Definition
1	Master console action
2	Master console information
3	Tape pool
4	Direct access pool
5	Tape library
6	Disk library
7	Unit record pool
8	Teleprocessing control
9	System security
10	System error/maintenance/system programmer information
11	Programmer information
12	Emulators
13-20	Reserved for customer use
21-28	Reserved for IBM- or customer-defined subsystem use
29-41	Reserved for IBM
42	General information about JES2 or JES3
43-64	Reserved for JES2 or JES3
65-96	Messages associated with particular processors
97-128	Messages associated with particular devices

If you omit the ROUTCDE, and CONSID or CONSNAME keywords, the system uses the routing code specified on the ROUTCODE keyword on the DEFAULT statement in the CONSOLxx member of SYS1.PARMLIB. See *z/OS MVS Initialization and Tuning Reference* for information about CONSOLxx.

,DESC=(descriptor code)

Specifies the message descriptor code or codes to be assigned to the message. Valid descriptor codes for the WTOR macro are:

- 7** Retain action message for life-of-task
- 13** Message previously automated

All WTOR messages are action messages that have an asterisk (*) sign displayed before the first character. This indicates a need for operator action.

The system adds descriptor code 7 to all WTOR messages. The system holds all WTOR messages until one of the following events occurs:

- The system deletes the WTOR message when the reply is received.
- You delete the WTOR message using the DOM macro. You should delete any unanswered WTOR messages that are no longer current.
- The system deletes the WTOR message at task termination.

The message processing facility (MPF) can suppress messages. For MPF to suppress messages, the hardcopy log must be active. The suppressed messages do not appear on any console; they do appear on the hardcopy log.

,MSGTYP=(msg type)

Specifies how the message is to be routed to consoles on which the MONITOR command is active. If you specify anything other than MSGTYP=N, which is the default, your message is routed according to your specification on MSGTYP, and the ROUTCDE parameter is ignored.

WTOR Macro

For SESS, JOBNAMEs, or STATUS, the message is to be routed to the console that issued the MONITOR SESS, MONITOR JOBNAMEs, or MONITOR STATUS command, respectively. When the message type is identified by the operating system, the message is routed to only those consoles that requested the information.

For Y or N, the message type describes what functions (MONITOR SESS, MONITOR JOBNAMEs, and MONITOR STATUS) are desired. N, or omission of the MSGTYP parameter, indicates that the message is to be routed as specified in the ROUTCDE parameter. Y creates an area in the WTO parameter list in which you can set message type information if you are coding a WTOR without any of the following parameters:

- KEY
- TOKEN
- CONSID
- CONSNAME
- TEXT
- RPLYISUR
- CART
- LINKAGE
- SYNCH

IBM recommends that you do not use MSGTYP=Y.

,MCSFLAG=(flag name)

Specifies one or more flag names whose meanings are shown below:

Table 44. MCSFLAG Flag Names

Flag Name	Meaning
REG0	Queue the message to the console whose console ID is passed in register 0. Issue the DISPLAY CONSOLES command to display valid console IDs. You can use register 0 to pass a 1-byte console ID (right-justified and padded to the left with zeroes) to identify the console to receive the message. However, IBM recommends you use the CONSID parameter instead of register 0.
RESP	The WTOR is an immediate command response.
REPLY	This is a reply to a WTOR.
BRDCST	Broadcast the message to all active consoles.
HRDCPY	Queue the message for hard copy only.
QREG0	Queue the message to the console whose console ID is passed in register 0. You can use register 0 to pass a 1-byte console ID (right-justified and padded to the left with zeroes) to identify the console to receive the message. However, IBM recommends you use the CONSID parameter instead of register 0.
NOTIME	Do not append time to the message.
NOCPY	Do not queue the message for hard copy.
CMD	The WTOR is a recording of a system command issued for hardcopy log purposes.

,RPLYISUR=reply console

Specifies a 12-byte field where the system will place the 8-byte console name and the 4-byte console ID of the console through which the operator replies to this message. When you specify this keyword in the list form, code it as RPLYISUR= with nothing after the equal sign.

,CONSID=console id

Specifies a 4-byte field containing the ID of the console to receive a message. Issue the DISPLAY CONSOLES command to display a list of valid console IDs. Use this ID in place of a console ID in register 0. If you specify a 4-byte console ID, or if you specify a console ID for an extended MCS console, you must use CONSID instead of register 0. If you specify a 1-byte console ID, you must right-justify it and pad to the left with zeroes.

Notes:

1. If you code the CONSID parameter using a register, the register must contain the console ID itself, rather than the address of the console ID.
2. When you code CONSID on the list form of WTOR, code it as CONSID= with nothing after the equal sign.
3. Do not use both CONSID and register 0 to pass a console ID, because the results are unpredictable. Be sure to clear the low-order byte of register 0 if you add the CONSID parameter to an existing invocation of WTOR.
4. CONSID is mutually exclusive with the CONSNAME parameter.

,CONSNAME=console name

Specifies an 8-byte field containing a 2- through 8-character name, left-justified and padded with blanks, of the console to receive a message. This parameter is mutually exclusive with the CONSID parameter. When you specify this keyword in the list form, code it as CONSNAME= with nothing after the equal sign. Do not use CONSNAME to pass a console name, together with register 0 to pass a console ID, because the results are unpredictable. Be sure to clear the low-order byte of register 0 if you add the CONSNAME parameter to an existing invocation of WTOR.

,SYSNAME=system name

Specifies an 8-byte input field containing a system name to be associated with this message. The system name is that of the system from which the caller issues the WTOR message. When you specify this parameter in the list form, code it as SYSNAME= with nothing after the equal sign.

,CART=cmd/resp token

Specifies an 8-byte field containing a command and response token to be associated with this message. You can specify any value as a command and response token. The command and response token is used to associate user information with a command and its command response. When you specify this keyword in the list form, code it as CART= with nothing after the equal sign.

,KEY=key

Specifies a field containing an 8-byte key to be associated with this message. The key must be EBCDIC if used with the MVS DISPLAY R command for retrieval purposes, but it must not be '*'. The key must be left-justified and padded on the right with blanks. If a register is used, it contains the address of the key. When this keyword is specified in the list form, it must be coded as KEY= with nothing after the equal sign.

,TOKEN=token

Specifies a field containing a 4-byte token to be associated with this message. This field is used to identify a group of messages that can be deleted by a DOM macro that includes TOKEN. The token must be unique within an address space, and can be any value. When you specify this keyword on the list form, code it as TOKEN= with nothing after the equal sign.

Note: When you code the TOKEN parameter using a register, the register must contain the token itself, rather than the address of the token.

WTOR Macro

,JOBID=*job id*

Specifies an 8-byte input field containing an ID that specifies the issuer of the WTOR message. When you specify this parameter in the list form, code it as **JOBID=** with nothing after the equal sign.

,JOBNAME=*jobname*

Specifies an 8-byte input field containing a name that specifies the issuer of the WTOR message. When you specify this parameter in the list form, code it as **JOBNAME=** with nothing after the equal sign.

,LINKAGE=SVC

,LINKAGE=BRANCH

Specifies how control is to pass to the WTOR service.

LINKAGE=SVC indicates the linkage is by a supervisor call. If **LINKAGE** is not specified, this is the default.

LINKAGE=BRANCH indicates the linkage is by a branch-and-link. You must use **SYNCH=YES** with this parameter. This parameter is used by programs that require the WTOR request to be handled synchronously.

When you specify this keyword in the list form, code it as **LINKAGE=** with nothing after the equal sign.

,SYNCH=NO

,SYNCH=YES

Specifies whether the WTOR request processes synchronously with the caller.

SYNCH=NO, the default, indicates that the request is not processed synchronously.

SYNCH=YES indicates the request is to be processed synchronously. This parameter is used in error and recovery environments, when normal message processing cannot be used. The message is sent to the console, and the reply is obtained immediately, before control is returned to the caller. Before return, the *reply* and *reply length* are moved to the areas specified by the caller, and the *ecb* marked "complete." Copies of the message and reply are queued for transcription to the hardcopy log.

If you specify **SYNCH=YES**:

- You must specify the parameter **LINKAGE=BRANCH**.
- Do not specify **MCSFLAG=HRDCPY**.

Your installation can determine which consoles can receive synchronous messages by using the **SYNCHDEST** parameter in the **CONSOLxx** member of **SYS1.PARMLIB**. For additional information on the **SYNCHDEST** parameter, see *z/OS MVS Initialization and Tuning Reference*.

ABEND Codes

WTOR might abnormally terminate with abend code X'D23'. See *z/OS MVS System Codes* for an explanation and programmer response for this code.

Return and Reason Codes

When the WTOR macro returns control to your program, GPR 15 contains one of the following hexadecimal return codes.

Table 45. Return Codes for the WTOR Macro

Return Code	Meaning and Action
00	<p>Meaning: Processing completed successfully.</p> <p>Action: None. Be sure to delete the message by issuing the DOM macro.</p>
04	<p>Meaning: Program error. One of the following occurred:</p> <ul style="list-style-type: none"> • The number of lines passed was 0; the request was ignored. • The message text length for a line was less than 1; all lines up to the error line were processed. <p>Action: Correct the problem and retry the request. If you used the TEXT parameter, make sure the parameter value and the data referenced are correct.</p>
0C	<p>Meaning: Program error. The line type was not valid. An end was forced at the point of the error unless the first line was an E line, in which case the request was ignored.</p> <p>Action: Correct the problem and retry the request.</p>
40	<p>Meaning: Environmental error. WTO was issued with LINKAGE=BRANCH; insufficient storage was available to queue the message for delayed issue. If SYNCH=NO was specified, the message was not queued for delayed issue. If SYNCH=YES was specified, the message was delivered for display, but not queued for hardcopy.</p> <p>Action: If you want the message to be delivered to the destination you requested, reissue the request. If the message was an action message that was not displayed, a DOM request is not required.</p>
44	<p>Meaning: Environmental error. WTO was issued with LINKAGE=BRANCH, SYNCH=YES; no usable console was available. The message was queued for hardcopy, but not delivered for display.</p> <p>Action: If you want the message to be delivered to the destination you requested, reissue the request. If the message was an action message that was not displayed, a DOM request is not required.</p>
48	<p>Meaning: Environmental error. WTO was issued with LINKAGE=BRANCH, SYNCH=YES; no usable console was available and insufficient storage was available to queue the message for delayed issue. The message was not delivered for display, nor queued for hardcopy.</p> <p>Action: If you want the message to be delivered to the destination you requested, reissue the request. If the message was an action message that was not displayed, a DOM request is not required.</p>
4C	<p>Meaning: Environmental error. WTO was issued with LINKAGE=BRANCH; no storage was available for the use of WTO processing. If WSPARM=0, no processing was done. If WSPARM does not equal zero, WTO loaded the wait state, but performed no other processing.</p> <p>Action: If you want the message to be delivered to the destination you requested, reissue the request. If the message was an action message that was not displayed, a DOM request is not required.</p>

Example 1

Issue a WTOR to the master console.

```
L8      EQU    8
      .
      .
      .
      WTOR    'USR902A REPLY YES OR NO TO CONTINUE.',REPLY,L8,REPECB, X
              ROUTCDE=(1),RPLYISUR=CONINFO
      .
      .
```

WTOR Macro

```
      .  
REPLY  DS    CL8  
REPECB DS    F  
CONINFO DS   CL12
```

Example 2

Issue a WTOR with the TEXT parameter. The message is to go to a specific console whose name is in field TOCON.

```
R4      EQU    4  
R5      EQU    5  
LENG12  EQU    12  
      .  
      .  
      .  
      LA      R4,CATMSG  
      LA      R5,TAPEAREA  
WTOR    TEXT=((R4),REPAREA,LENG12,TAPEECB),  
        CONSNAME=TOCON,  
        RPLYISUR=(R5) X  
      .  
      .  
      .  
CATMSG  DC     AL2(L'REP64) 00011800  
REP64   DC     C'USR922A INDICATE NUMBER OF TAPE DRIVES REQUIRED.'  
TOCON   DC     CL8'TAPECON '  
REPAREA DS     CL12  
TAPEECB DS     F  
TAPEAREA DS    CL12
```

Example 3

Issue a branch-entry WTOR.

```
C80     EQU    80  
      .  
      .  
      .  
WTOR    'USR940I ENTER THE NAMES OF AFFECTED JOBS:',REPAR6,C80,JX  
        OBSECB,RPLYISUR-JOBCONS, X  
        ROUTCDE=(1),LINKAGE=BRANCH,SYNCH=YES  
      .  
      .  
      .  
REPAR6  DS     CL80  
JOBSECB DS     F  
JOBCONS DS     CL12
```

Example 4

Issue a WTOR using the TEXT parameter with the list and execute forms of the macro. The console ID to which the message is to be queued is assumed to be in field MYCONID. On the TEXT parameter for the execute form, commas mark the positions of *reply addr* and *ecb addr*; for the list form, a comma marks the position of *reply length*.

```
R12     EQU    12  
C50     EQU    50  
      USING *,R12  
      .  
      .  
      .  
WTOR    MF=(E,M2,EXTENDED),TEXT=(MESSAGE,,C50,),CONSID=MYCONID, X  
        RPLYISUR=MYCONAR  
      .  
      .
```

```

M2      .
        DS      0H
        WTOR    TEXT=(,RAREA,,MYECB),CONSID=,ROUTCDE=(2),RPLYISUR=,MF=L
MYCONID DS      F
RAREA   DS      CL50
MYECB   DS      F
MYCONAR DS      CL12
MESSAGE DC      AL2(L'MTEXT)
MTEXT   DC      C'USR930A REQUEST IS AMBIGUOUS. RESPECIFY DEVICE.'
        END
```

WTOR—List Form

Use the list form of the WTOR macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to store the parameters.

Syntax

The list form of the WTOR macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede WTOR.
WTOR	
b	One or more blanks must follow WTOR.

'msg',reply addr,reply length,ecb addr
('text',reply addr,reply length,ecb addr)
(('text',line type,...,'text',line type),reply addr,reply length,ecb addr)
TEXT=(,reply addr,reply length,ecb addr)
TEXT=(((,line type),(,line type),..., (,line type)),reply addr,reply length,ecb addr)
msg: Up to 122 characters.
text: Up to 122 characters.
reply addr: A-type address.
reply length: Symbol, decimal digit. The minimum length is 1; the maximum length is 119.
ecb addr: A-type address.
The permissible *line types*, text lengths, and maximum numbers are shown below:

line type	text	maximum number
C	31 char	1 C type
L	67 char	2 L type
D	67 char	14 D type
DE	67 char	1 DE type
	or	
E	None	1 E type

The maximum total of lines that can be coded in one instruction is 14.

WTOR Macro

Notes:

1. If you code *'msg'...* or *('text'...)*, it must be the first parameter you code.
2. If you do not code *reply addr* on the list form of WTOR, mark its position with a comma, and code *reply addr* on the execute form. The same is true for *reply length* and *ecb addr*.

,ROUTCDE=(*routing code*)

routing code: Decimal digit from 1 to 128. The *routing code* is one or more codes, separated by commas, or a hyphen to indicate a range.

,DESC=(*descriptor code*)

descriptor code: Decimal number 7 or 13. If you code both 7 and 13, separate them with commas.

,MSGTYP=(*msg type*)

msg type: Any of the following:

N	SESS,JOBNAMES
Y	SESS,STATUS
SESS	JOBNAMES,STATUS
JOBNAMES	SESS,JOBNAMES,STATUS
STATUS	

,MCSFLAG=(*flag name*)

flag name: Any combination of the following, separated by commas:

REG0	HRDCPY
RESP	QREG0
REPLY	NOTIME
BRDCST	NOCOPY
CMD	

,RPLYISUR=

Parameter value not required for list form. Code only,RPLYISUR=. If you code RPLYISUR on the list form of WTOR, you must code RPLYISUR on the execute form.

,CONSID=

Parameter value not required for list form. Code only,CONSID= (or ,CONSNAME=). If you code CONSID (or CONSNAME) on the list form of WTOR, you must code CONSID (or CONSNAME) on the execute form.

,CONSNAME=

,SYSNAME=

Parameter value not required for list form. Code only,SYSNAME=. If you code SYSNAME on the list form of WTOR, you must code SYSNAME on the execute form.

,CART=

Parameter value not required for list form. Code only,CART=. If you code CART on the list form of WTOR, you must code CART on the execute form.

,KEY=

Parameter value not required for list form. Code only,KEY=. If you code KEY on the list form of WTOR, you must code KEY on the execute form.

,TOKEN=

Parameter value not required for list form. Code only,TOKEN=. If you code TOKEN on the list form of WTOR, you must code TOKEN on the execute form.

,JOBID=

Parameter value not required for list form. Code only,JOBID=. If you code JOBID on the list form of WTOR, you must code JOBID on the execute form.

,JOBNAME=

Parameter value not required for list form. Code only,JOBNAME=. If you code JOBNAME on the list form of WTOR, you must code JOBNAME on the execute form.

,LINKAGE=	Parameter value not required for list form. Code only,LINKAGE=. If you code LINKAGE on the list form of WTOR, you must code LINKAGE on the execute form.
,SYNCH=NO ,SYNCH=YES	Default: NO
,MF=L	

Parameters

The parameters are explained under the standard form of the WTOR macro with the following exception:

,MF=L
Specifies the list form of the WTOR macro.

WTOR—Execute Form

Use the execute form of the WTOR macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

The message cannot be modified on the execute form of the macro if you code inline text ('msg'... or ('text'...)) on the list form.

Syntax

The execute form of the WTOR macro is written as follows:

<i>name</i>	<i>name</i> : Symbol. Begin <i>name</i> in column 1.
b	One or more blanks must precede WTOR.
WTOR	
b	One or more blanks must follow WTOR.

,reply addr,reply length,ecb addr
TEXT=(text addr,reply addr,reply length,ecb addr)
TEXT=(((text addr,),(text addr,),...(text addr,)),reply addr,reply length,ecb addr)
reply addr: RX-type address, or register (2) - (12).
reply length: Symbol, decimal digit, or register (2) - (12).
The minimum length is 1; the maximum length is 119.
ecb addr: RX-type address, or register (2) - (12).
text addr: RX-type address or register (2) - (12).

WTOR Macro

Notes:

1. If you code *reply addr*, *reply length*, *ecb addr*, it must be the first parameter you code and must be preceded by a comma.
2. If you specify inline text on the list form ('*msg'*... or ('*text'*...)), do not code the TEXT keyword on the execute form.
3. If you do not code *reply addr* on the execute form of WTOR, mark its position with a comma, and code *reply addr* on the list form. The same is true for *reply length* and *ecb addr*.

,RPLYISUR=*reply console*

reply console: RX-type address or register (2) - (12). If you code RPLYISUR on the execute form of WTOR, you must code RPLYISUR on the list form.

,CONSID=*console id*

,CONSNAME=*console name*

console id: RX-type address or register (2) - (12).

console name: RX-type address or register (2) - (12).

If you code CONSID (or CONSNAME) on the execute form of WTOR, you must code the same parameter on the list form.

,SYSNAME=*system name*

system name: RX-type address or register (2) - (12).

If you code SYSNAME on the execute form of WTOR, you must code the same parameter on the list form.

,CART=*cmd/resp token*

cmd/resp token: RX-type address or register (2) - (12).

If you code CART on the execute form of WTOR, you must code CART on the list form.

,KEY=*key addr*

key addr: RX-type address or register (2) - (12).

If you code KEY on the execute form of WTOR, you must code KEY on the list form.

,TOKEN=*token addr*

token addr: RX-type address or register (2) - (12).

If you code TOKEN on the execute form of WTOR, you must code TOKEN on the list form.

,JOBID=*job id*

job id: RX-type address or register (2) - (12).

If you code JOBID on the execute form of WTOR, you must code JOBID on the list form.

,JOBNAME=*jobname*

jobname: RX-type address or register (2) - (12).

If you code JOBNAME on the execute form of WTOR, you must code JOBNAME on the list form.

,LINKAGE=SVC

,LINKAGE=BRANCH

Default: SVC

If you code LINKAGE on the execute form of WTOR, you must code LINKAGE on the list form.

,SYNCH=NO

,SYNCH=YES

Default: NO

,MF=(E,*list addr*)

,MF=(E,*list addr*,EXTENDED)

list addr: RX-type address, or register (1) - (12).

Parameters

These parameters are explained under the standard form of the WTOR macro, with the following exceptions:

,reply addr,reply length,ecb addr

If you code *reply addr,reply length,ecb addr*, it must be the first parameter you code and must be preceded by a comma.

,MF=(E,list addr)

,MF=(E,list addr,EXTENDED)

Specifies the execute form of the WTOR macro.

list addr specifies the area that the system uses to store the parameters.

If you specify *reply addr*, *reply length*, *ecb addr*, or RPLYISUR on the execute form of WTOR, together with one or more of the following parameters, you must specify EXTENDED for the system to generate the parameter list correctly:

- KEY
- TOKEN
- CONSID
- CONSNAME
- TEXT
- RPLYISUR
- CART
- LINKAGE
- SYNCH

Example

Write a message with a prebuilt parameter list pointed to by register 1.

```
WTOR      MF=(E,(1))
```

Appendix. Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen-readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

Using assistive technologies

Assistive technology products, such as screen-readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using it to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to *z/OS TSO/E Primer*, *z/OS TSO/E User's Guide*, and *z/OS ISPF User's Guide Volume I* for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

Notices

This information was developed for products and services offered in the USA.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming Interface Information

This book is intended to help the customer to code macros that are available to authorized assembler language programs. This book documents intended Programming Interfaces that allow the customer to write programs to obtain services of z/OS.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

- AFP
- CICS
- DFSORT
- ESA/390
- IBM
- IBMLink
- MVS/ESA
- MVS/SP
- MVS/XA
- OS/390
- Print Services Facility (PSF)
- RMF
- Resource Link
- SP
- SP1
- SP2
- z/Architecture
- z/OS
- z/OS.e

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product and service names may be the trademarks or service marks of others.

Index

A

- accessibility 357
- addressing mode and the services 2
- ALET qualification
 - of parameters 3
- AR () mode
 - description 3
- ASC (address space control) mode
 - defining 3
- authorization
 - testing caller 199

C

- callable service
 - coding 15
- caller
 - testing authorization 199
- coding the callable services 15
- coding the macros 12
- continuation line 15
- control access to a serially reusable resource 31
- CPU time
 - obtaining accumulated 203

D

- disability 357
- documents, licensed x

E

- event
 - waiting for one or more 305

F

- functional recovery routine
 - setting up 25

I

- IHATRBPL mapping macro 173
- IHATREPL mapping macro 174

K

- keyboard 357

L

- licensed documents x
- log
 - writing 311
- LookAt message retrieval tool x

M

- macro
 - coding 12
 - forms 11
 - level
 - selecting 1
 - sample 13
 - selecting level 1
 - user parameter, passing 4
 - X-macros
 - using 10
- message retrieval tool, LookAt x

N

- Notices 359

P

- parameter
 - setting return 43
- process
 - putting in process-must-complete mode 103
- process-must-complete mode 103

Q

- query virtual server 185
- QVS 185

R

- RETRIEVE service
 - reason codes 155
 - return codes 155

S

- service
 - ALET qualification 3
 - summary 16
- services
 - addressing mode 2
 - ASC mode
 - defining 3
 - using 1
- SETFRR macro 25
- SETLOCK macro 31
- SETRP macro 43
- shortcut keys 357
- SJFREQ macro 53
 - RETRIEVE service 55, 56
 - SWBTU_MERGE service 55, 60, 61
 - TERMINATE service 55, 82
 - VERIFY service 55, 69
- SPIE macro 87

- SPOST macro 93
- SRB (service request block)
 - transferring control 197
- SRB status 95
- SRBSTAT macro 95
- SRBTIMER macro 99
- STATUS macro 103
- storage
 - obtaining and releasing 109
- STORAGE macro 109
- subtask
 - starting and stopping 103
- SUSPEND macro for RBs 127, 129
- SUSPEND macro for SRBs 129
- SVC exit
 - type 6 207
- SVCUPDTE macro 135
- SWA manager
 - invoking in locate mode 145
- SWAREQ macro 145
- SWBTUREQ macro 151
- SYNCH and SYNCHX macros 161, 169
- synchronous exit
 - to a processing program 161
- SYSEVENT macro 169

T

- T6EXIT macro 207
- TCBTOKEN macro 189
 - description 189
- TCTL macro 197
- TESTAUTH macro 199
- time limit
 - establishing for system service 99
- TIMEUSED macro 203

U

- UCB (unit control block)
 - obtaining address 247
 - pinning 257
 - scanning 265
 - unpinning 257
- UCBINFO macro 209
- UCBLOOK macro 247
- UCBPIN macro 257
- UCBSCAN macro 265
- user parameter
 - passing 4

V

- vector time
 - obtaining accumulated 203
- virtual storage
 - map 287
 - obtaining private area region size 301
 - verifying allocation 295
- VSMLIST macro 287
- VSMLOC macro 295

- VSMREGN macro 301

W

- WAIT macro 305
- WTL macro 311
- WTO macro 319
- WTOR macro 339

X

- X-macros
 - using 10

Readers' Comments — We'd Like to Hear from You

z/OS
MVS Programming: Authorized
Assembler Services Reference, Volume 4
(SETFRR-WTOR)

Publication No. SA22-7612-02

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Complete	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to find	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to understand	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Well organized	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Applicable to your tasks	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name

Address

Company or Organization

Phone No.



Cut or Fold
Along Line

Fold and Tape

Please do not staple

Fold and Tape



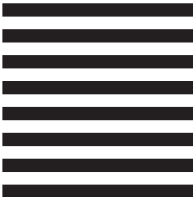
NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie, NY
12601-5400



Fold and Tape

Please do not staple

Fold and Tape

Cut or Fold
Along Line



Program Number: 5694-A01, 5655-G52

Printed in U.S.A.

SA22-7612-02

